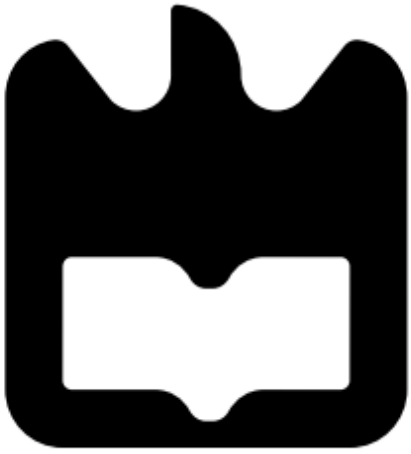
Tiago Ferreira Martins

Processamento Analítico de Dados em Aplicações de Monitorização de Performance de Redes utilizando In-Memory Data Grids

Analytical Data Processing in Network Performance Monitoring Applications using In-Memory Data Grids



Tiago Ferreira Martins

Processamento Analítico de Dados em Aplicações de Monitorização de Performance de Redes utilizando In-Memory Data Grids

Analytical Data Processing in Network Performance Monitoring Applications using In-Memory Data Grids

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor José Manuel Matos Moreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Aníbal Manuel de Oliveira Duarte
Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
(Presidente)

vogais / examiners committee

Prof. Doutor Alexandre Miguel Barbosa Valle de Carvalho
Professor Auxiliar da Faculdade de Engenharia do Porto
(Arguente Principal)

Prof. Doutor José Manuel Matos Moreira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
(Orientador)

agradecimentos

Agradeço, em particular, ao meu orientador José Manuel Matos Moreira, pelo profissionalismo, dedicação, apoio e incentivo constante no desenvolvimento do projeto de dissertação.

Ao Centro de Investigação Nokia Solutions and Networks Aveiro, pelo acolhimento nas instalações e disponibilização dos recursos necessários para o desenvolvimento do trabalho.

Ao Engenheiro Paulo Costa e Engenheiro Ricardo Marques, supervisores do projeto no Centro de Investigação, pela amizade, confiança e ajuda demonstrada em todo o processo.

Um agradecimento muito especial aos familiares e amigos pelo apoio incondicional.

palavras-chave

big data, in-memory data grids, Nokia Performance Manager, Apache Ignite/GridGain, Apache Geode/Pivotal Gemfire.

resumo

Nos últimos anos, assiste-se a um forte incremento no que diz respeito ao volume de dados e sua respetiva valorização por parte das organizações, sobretudo em aplicações de monitorização de performance de redes, como se observa na ferramenta Nokia Performance Manager (NPM).

A situação atual justifica-se com o crescimento do setor das telecomunicações, com a necessidade de dar resposta à constante evolução das tecnologias de rede 2G, 3G e atualmente o 4G ou LTE, no processo de centralização, processamento e armazenamento de dados ou indicadores de performance de rede recolhidos por antenas espalhadas em diversas zonas geográficas. Neste contexto, a ferramenta NPM disponibiliza ao cliente uma visão global do desempenho da rede em tempo real, através da geração de relatórios pré-configurados e posterior armazenamento para futuras visualizações. Perante a necessidade de armazenar esses relatórios por um período mais alargado, sem limitações de armazenamento e de forma mais eficiente verifica-se que os tradicionais sistemas de gestão e armazenamento, os RDBMS em disco e cache, não respondem da melhor forma a estas necessidades.

Posto isto, surgiu o conceito *"In-Memory Data Grid"* como uma plataforma de dados completamente distribuídos sobre uma arquitetura em memória. De modo a provar este conceito foi implementado um protótipo de *software* para o processo de *caching* de relatórios, funcionalidade do NPM, utilizando as tecnologias Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire e posteriormente foi realizado um ambiente de testes que permitiu avaliar as duas soluções e determinar aquela que seria a solução ideal a integrar na ferramenta NPM.

keywords big data, in-memory data grids, Nokia Performance Manager, Apache Ignite/GridGain, Apache Geode/Pivotal Gemfire.

abstract In recent years, there has been a strong increase in the volume of data and its respective valuation by organizations, especially in network performance monitoring applications, as seen in the Nokia Performance Manager (NPM) tool.

The current situation is justified by the growth of the telecommunications sector, with the need to respond to the constant evolution of 2G, 3G and currently 4G or LTE network technologies, in the process of centralizing, processing and storing data or indicators of Network performance collected by distributed antennas in different geographic areas. In this context, the NPM tool provides the customer with a global view of the network performance in real time, through the generation of pre-configured reports and later storage for future visualizations. Given the need to store these reports for a longer period, without storage limitations, and more efficiently, traditional storage and management systems, RDBMS on disk and cache, do not best meet these needs.

That said, the "In-Memory Data Grid" concept emerged as a completely distributed data platform on an in-memory architecture. In order to prove this concept, a software prototype was implemented for the report caching process, NPM functionality, using the Apache Ignite/GridGain and Apache Geode/Pivotal Gemfire technologies, and later a test environment was carried out to evaluate the two Solutions and determine what would be the ideal solution to integrate into the NPM tool.

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

Acrónimos

API	Interface de Programação de Aplicações
ACID	Atomicidade, Consistência, Isolamento, Durabilidade
BASE	Basicamente Disponível, Estado Leve, Eventualmente Consistente
BI	Inteligência Empresarial
CSV	Valores Separados por Vírgulas
CLI	Interface Linha de Comandos
CPU	Unidade Central de Processamento
CAP	Consistência, Disponibilidade, Tolerância a Falhas
DW	Armazém de Dados
DBMS	Sistema de Gestão de Bases de Dados
ETL	Extração, Transformação, Carga
GPS	Sistema de Posicionamento Global
HPC	Computação de Alto Desempenho
HA	Alta Disponibilidade
HDFS	Sistema de Arquivos Distribuídos Hadoop
HDD	Disco Rígido
IMDG	Grid de Dados em Memória
IMDB	Sistema de Gestão de Bases de Dados
IMC	Computação em Memória
IMCG	Computação Grid em Memória
IMDBMS	Sistema de Gestão de Bases de Dados em Memória
IoT	Internet das Coisas
IP	Protocolo de Internet
JSON	*JavaScript Object Notation*

JVM	Máquina Virtual Java
KPI	Indicador Chave de Desempenho
LTE	*Long-Term Evolution*
LNBTS	*Long-Term Evolution Node Base Transceiver Station*
LNCELL	*Long-Term Evolution Node Cell*
MRAM	*Magneto Resistive Random-Access Memory*
MRBTS	*Multi Radio Base Transceiver Station*
MPP	Processamento Paralelo Massivo
NSN	*Nokia Solutions and Networks*
NPM	*Nokia Performance Manager*
NoSQL	Linguagem de Consulta Estruturada Não Somente SQL
New SQL	Nova Linguagem de Consulta Estruturada
OLAP	Processamento Analítico Online
OLTP	Processamento de Transações em Tempo Real
OPEX	Despesas Operacionais
OQL	*Object Query Language*
OMeS	*Open Measurement Standard*
PLMN	*Public Land Mobile Network*
QoS	Qualidade de Serviço
RDBMS	Sistema de Gestão de Bases de Dados Relacionais
RAM	Memória de Acesso Aleatório
SQL	Linguagem de Consulta Estruturada
SSD	Unidade de Estado Sólido
TPS	Transações por Segundo
UI	Interface de Utilizador
URL	Localizador Padrão de Recursos
XML	*Extensible Markup Language*
3GPP	*3rd Generation Partnership Project*

Capítulo 1

Introdução

"You can have data without information, but you cannot have information without data." – Daniel Keys Moran.

Em plena *"Era da Informação"* assiste-se a um aumento do volume de dados e sua respetiva valorização por parte das organizações, bem como a um aumento das fontes que lhe dão origem, nomeadamente, a internet, telecomunicações, redes sociais *online*, imagens digitais, vídeos, mensagens de texto ou dados de sensores, entre outras. Este fenómeno deve-se à constante evolução tecnológica e humana, ao crescimento do mercado de computadores e dispositivos móveis, à facilidade de acesso à internet e sobretudo à interação com a tecnologia, que se verifica atualmente.

Os *RDBMS* (Sistemas de Gestão de Bases de Dados Relacionais) têm sido utilizados pela maioria das organizações no processo de armazenamento e acesso a dados através de modelos relacionais. Estes oferecem um modelo simples de utilização, propriedades *ACID* (Atomicidade, Consistência, Isolamento, Durabilidade) no que respeita ao processamento de transações e o *SQL* (Linguagem de Consulta Estruturada), que é uma linguagem declarativa que permite fazer consultas em bases de dados relacionais.

No entanto, atualmente, a informação torna-se cada vez menos estruturada e afasta-se do modelo relacional tradicional, conduzindo-nos ao conceito *Big Data*. Este descreve o conjunto enorme de dados, estruturados e não estruturados, que são reunidos e analisados tendo em vista a tomada de decisões. A sua utilização está na base da competição e crescimento a que se assiste no âmbito das organizações, criando valor próprio, reduzindo os desperdícios e aumentando a

qualidade dos produtos e serviços bem como a sua produtividade. Perante esta evolução, as referidas organizações têm que se adaptar a novas metodologias e evoluir para tecnologias que oferecem grande capacidade de armazenamento de dados, provenientes de diversas fontes, com um processamento de alta performance e que ajudam no processo de transformação de dados em informação ou conhecimento útil em diversos setores, nomeadamente o comercial, financeiro e social.

O *Big Data* está a dar origem a novas tecnologias de bases de dados, tais como: o *NewSQL* (Nova Linguagem de Consulta Estruturada), o *NoSQL* (Linguagem de Consulta Estruturada Não Somente SQL) e o *IMDBMS* (Sistemas de Gestão de Bases de Dados em Memória).

O *NewSQL*, uma evolução dos tradicionais *RDBMS*, apresenta as mesmas características ao nível de linguagem *SQL*, propriedades *ACID* e modelação relacional, no entanto, oferece um sistema distribuído procurando obter melhor desempenho, alta disponibilidade e escalabilidade horizontal. Atendendo a que o *NewSQL* não permite armazenar informação não estruturada, surge o *NoSQL* que oferece diversos tipos de *DBMS* (Sistemas de Gestão de Bases de Dados) para o armazenamento e acesso a dados não estruturados, como por exemplo, *DBMS* baseados em pares Chave / Valor; *DBMS* orientado a documentos; *DBMS* orientado a colunas e *DBMS* baseados em grafos, apresentando agora, uma escalabilidade horizontal que permitem utilizar esquemas semiestruturados e não-estruturados. Tratando-se de sistemas não-relacionais, estas soluções, no entanto, nem sempre suportam a linguagem *SQL*, dificultando a compatibilidade com os sistemas legados. Outro problema desta tecnologia é a consistência dos dados, pois não garantem a totalidade das propriedades *ACID*, emergindo um novo conceito, as propriedades *BASE* (Basicamente Disponível, Estado Leve, Eventualmente Consistente*)*, daí a conhecida expressão, agora citada: *"bases de dados NoSQL ou não terá disponibilidade ou não terá consistência"*, isto é, pelo teorema de CAP apenas são garantidas duas das três propriedades enunciadas no mesmo (Souza 2014).

Neste contexto, surgiram as *IMDBMS*, que se caracterizam pela utilização da memória RAM para o armazenamento e acesso de dados, ao invés da utilização do disco rígido resolvendo os problemas de consistência, através das propriedades ACID, que permitem o armazenamento de dados estruturados e não estruturados e que oferecem escalabilidade horizontal, sistema

distribuído e alto desempenho, com recurso total à memória. O problema deste tipo de bases de dados em memória é carecer de capacidade de persistência (Plattner 2009).

Para colmatar a capacidade de persistência emergiram as tecnologias de *grid* em memória, mais conhecidas por *IMDG* (*Grid* de Dados em Memória) e que se caracterizam pelo armazenamento e processamento de dados em memória *RAM* e pela distribuição dos mesmos por diversas máquinas distribuídas, garantindo escalabilidade e alta disponibilidade. Temos então um sistema dinâmico distribuído com alta performance pela rapidez de acesso aos dados e ainda todas as características das bases de dados tradicionais.

Nos últimos tempos assiste-se a um grande esforço para conciliar uma maior capacidade de armazenamento com o alto processamento de dados de forma rápida. Neste sentido, foram desenvolvidos sistemas, como o *Impala*, *Storm*, *HAWQ*, *Spark*, *Map Reduce,* que apesar de funcionarem de forma independente, oferecem um processamento avançado e que consequentemente conduziu ao desenvolvimento de uma plataforma única, as tecnologias *IMDG*.

Esta dissertação centra-se na utilização de tecnologias *IMDG* sendo que a proposta de projeto foi apresentada em parceria com o Centro Tecnológico da NSN (Nokia Solutions and Networks). Tem como objetivo desenvolver um estudo experimental sobre a utilização de soluções baseadas na tecnologia *IMDG*, centrada na funcionalidade de *caching* de relatórios de rede na ferramenta *NPM* (*Nokia Performance Manager*), de forma a permitir aos operadores de telecomunicações monitorizar a performance ou comportamento da rede, analisando as causas em caso de degradação do desempenho dos serviços. Este tipo de ferramentas geram dados ou indicadores de performance em redes que utilizam tecnologias 2G, 3G e a evolução para LTE ou 4G, permitindo fazer posteriormente uma análise do fluxo de dados em tempo real.

1.1. Nokia Solutions and Networks (NSN)

A Nokia Solutions and Networks (NSN), subsidiária integral da Nokia Corporation fundada em 2007, é uma empresa multinacional de redes de dados e equipamentos de telecomunicações sediada na cidade de Espoo, na Finlândia.

A empresa opera em mais de 150 países e conta com 60.000 colaboradores em todo o mundo, sendo que, em Portugal, colaboram aproximadamente 2000 pessoas. Apresenta-se como líder de mercado em equipamentos, soluções e serviços para redes de banda larga móvel.

O Centro Tecnológico da NSN localizado em Lisboa e em Aveiro reúne um conjunto de profissionais dedicados à atividade de investigação e promove a cooperação entre a indústria e o mundo académico, tornando Portugal um dos maiores centros de inovação e de exportação de conhecimento para todo o mundo.

O primeiro contacto com a Nokia Solutions and Networks ficou a dever-se à escolha da tese de dissertação e ao interesse em trabalhar na área do *Big Data*, surgindo neste contexto a oportunidade de contactar com uma empresa mundialmente conhecida e com uma realidade completamente diferente do mundo universitário, num projeto aliciante e que futuramente pode ser utilizado como uma nova forma das empresas armazenarem e gerirem grandes volumes de dados.

1.2. Nokia Performance Manager (NPM)

Os Sistemas de Gestão da Performance têm um papel fundamental ao nível das operações de uma rede de telecomunicações, pois lidam com o crescente volume e complexidade de dados provenientes das redes de pequena e larga escala. Estes reúnem todo o volume de dados de diversas fontes e transformam-nos posteriormente em informação, centralizando-os e apresentando uma visão imediata e unificada da rede, permitindo que os problemas sejam detetados com antecedência. Desta forma, proporcionam, uma otimização de recursos e processos que se reflete numa eficiência operacional e consequentemente apresentam vantagens na qualidade de serviço (QoS) que fornecem ao cliente bem como vantagens ao nível dos negócios.

Um estudo realizado pela NSN através da análise do desempenho de uma rede de telecomunicações, derivada de relatórios elaborados, conclui que este tipo de sistemas permitem uma redução significativa do OPEX (Despesas Operacionais) entre 10% a 15% no tempo de planeamento de otimização de uma rede (N. Networks 2014) .

A ferramenta NPM é um exemplo deste sistema pois recorre ao ORACLE para gerir o armazém de dados centralizado e disponibiliza ao cliente, uma visão global do desempenho e

estado da rede em tempo real. Através desta ferramenta temos acesso a um conjunto de funcionalidades, tais como: visualização de relatórios em tabelas e gráficos; documentação de relatórios que incluem as fórmulas KPI (Indicador-Chave de Desempenho); exportação para Excel e CSV (Valores Separados por Vírgulas) com vistas em tabelas e gráficos; relatórios de navegação; agendamento de relatórios; filtragem e pesquisa avançada, entre outras, como se observa na Figura 1 (N. S. and Networks 2014).

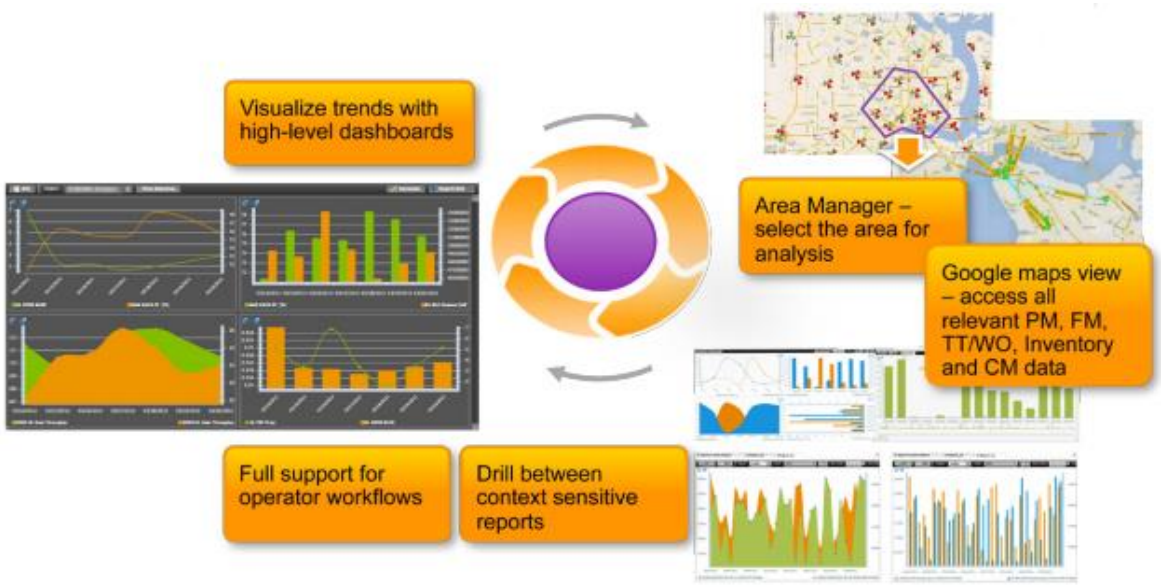


Figura 1 - Visão Geral Nokia Performance Manager (NPM), retirado de (Network 2012).

1.3. Formulação do Problema

O desenvolvimento do estudo foca-se essencialmente na funcionalidade de *caching* de relatórios de rede da ferramenta NPM. Esta permite gerar relatórios pré-configurados de indicadores de desempenho da rede, através de filtros avançados, recorrendo a dados armazenados na *Database ORACLE*, que posteriormente são armazenados na *cache* para futuras requisições do mesmo relatório. O fluxo deste processo requer que seja sempre verificada a existência do relatório requisitado em cache, e caso não exista proceder à sua geração e respetivo armazenamento. Isto porque o tempo gasto no processo de geração de um relatório é bastante superior ao tempo de devolver um relatório que já foi gerado e que está armazenado na cache, o que se explica pela performance significativa na utilização da memória, ao invés do disco.

No entanto, a funcionalidade atualmente utilizada pela ferramenta NPM apresenta-se como sendo uma solução experimental sobre tecnologia *Database ORACLE* que a longo prazo pode ser um pouco limitada face às tecnologias emergentes, as IMDGs. Estamos a falar de uma ferramenta (NPM) que regista informação de aparelhos de rede e que processa cerca de 200 milhões de registos por hora em redes médias, com tendência para chegar a um bilião a curto prazo. Tendo mecanismos fortemente intensivos ao nível de I/O (leituras e escritas), fazem com que as metodologias de *caching* ao nível de processamento e armazenamento sejam de extrema importância.

Neste sentido, o estudo desenvolvido nesta dissertação tem o intuito de substituir a curto prazo, o sistema de *caching* de relatórios de rede, atualmente fornecido pela *Database ORACLE*, por um sistema completamente distribuído em memória, com escalabilidade, performance e alta disponibilidade, usando IMDGs.

1.4. Objetivos

Perante o problema apresentado relativamente à funcionalidade de *caching* de relatórios na ferramenta NPM, foi proposto estudar a utilização de tecnologias IMDG direcionadas para o uso da memória na tarefa de armazenar e processar relatórios com indicadores provenientes da monitorização de performance de redes. O estudo experimental desenvolveu-se em quatro fases distintas:

- Seleção de uma funcionalidade ou processo da plataforma NPM;
- Desenho e implementação de um protótipo do sistema para o mesmo, utilizando dois produtos que utilizem tecnologias IMDG;
- Preparação de um ambiente de testes e avaliação dos produtos numa arquitetura real;
- Recolha, análise e discussão dos resultados aos testes realizados;

Após uma análise dos produtos que utilizam as tecnologias IMDG, foram selecionadas para o desenvolvimento deste trabalho os seguintes: Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire.

A estratégia de desenvolvimento do sistema para o processo selecionado na plataforma NPM prende-se com a construção de uma arquitetura Cliente-Servidor com recurso à virtualização de máquinas. Cada máquina é observada como um nó de um *cluster* e o *software* desenvolvido é configurado em cada um dos nós existentes, para que servidores e clientes comuniquem de forma interligada. Para isto, procedeu-se à configuração da topologia adequada nos servidores de *cache* e respetiva estruturação do modelo de dados, bem como a implementação de uma interface de cliente Java para simular um ambiente de *caching* de relatórios de rede na *grid* de dados em memória, utilizando os produtos mencionados. Estes relatórios são gerados com recurso a dados agregados numa base de dados e a mecanismos de consulta implementados na própria interface, sendo estes distribuídos posteriormente nos servidores de cache existentes sobre a estrutura de dados implementada.

De seguida, foi preparado um ambiente de testes de carga numa arquitetura real, configurada e desenvolvida na Nokia, utilizando uma ferramenta de recolha de métricas, denominada por Yardstick. O objetivo é testar ações de geração e armazenamento de relatórios de rede nos servidores de cache (PUT) e obtenção desses relatórios a partir dos servidores de cache (GET) de forma simultânea num prazo de tempo definido. Com isto, pretendemos testar o sistema IMDG em memória implementado para os dois produtos selecionados e proceder à recolha e análise de resultados, com o intuito de mostrar aquela que seria uma solução viável numa possível remodelação da funcionalidade na plataforma NPM.

1.5. Estrutura da Dissertação

O estudo desenvolvido será apresentado ao longo de cinco capítulos, que se descrevem, em seguida, de forma sucinta.

Inicialmente, num capítulo introdutório, cumpre elaborar uma breve contextualização do projeto, descrevendo a empresa e a ferramenta NPM que serviu de base ao mesmo, e ainda apresentar o problema e os objetivos que procuramos desenvolver.

Em seguida, num capítulo que dominaremos de "Sistemas de Gestão e Armazenamento de Dados", aborda-se o conceito *Big Data* e o crescimento inerente dos três tipos de DBMS, o NewSQL, o NoSQL e as IMDBMS como sistemas de armazenamento de informação. Destaque

para a tecnologia IMDG, a base deste estudo experimental por ter sido a solução escolhida para o desenvolvimento do sistema distribuído de armazenamento e processamento de dados.

Num terceiro capítulo, "Arquitetura e Implementação" temos como ponto fulcral demonstrar o estado atual da plataforma NPM na funcionalidade de *caching* de relatórios, no que respeita à implementação da cache ORACLE, o modelo 3GPP e interface de utilizador e ainda, de forma detalhada apresentar a nova solução com IMDGs ao nível do diagrama de classes, estrutura de dados, mecanismos de consulta, arquitetura do sistema e configurações de *software* para o protótipo implementado.

O capítulo 4, "Ambiente de Testes", apresenta como foi preparado o ambiente de testes numa arquitetura real na Nokia, bem como toda a metodologia desenvolvida, por forma a recolher, analisar e discutir os resultados obtidos.

Por fim, cumpre elaborar uma reflexão do trabalho desenvolvido, em molde conclusivo, num quinto capítulo.

Capítulo 2

Sistemas de Gestão e Armazenamento de Dados

Este capítulo centra-se no estado da arte em vários tópicos relacionados com esta dissertação. Está dividida da seguinte forma: A secção 2.1 apresenta o conceito de *Big Data* descrevendo o seu crescimento e importância nos dias de hoje. Em seguida é feita uma apresentação dos principais paradigmas emergentes nesta área, relacionadas com bases de dados, nomeadamente, a secção 2.2 que descreve o paradigma emergente NewSQL, a secção 2.3 que apresenta o conceito de NoSQL e os vários tipos de bases de dados inerentes, e a secção 2.4, que retrata a evolução para bases de dados que armazenam e acedem a dados com recurso à memória, ao invés do disco rígido, com as IMDBMS. Por ser a tecnologia usada no trabalho, dá-se ênfase na secção 2.5 às IMDGs e às propriedades que a caracterizam. No seguimento da análise detalhada desta tecnologia, recorreu-se ao estudo da documentação dos produtos existentes no mercado das IMDGs com o intuito de selecionar os que mais se adequam ao trabalho que se pretende desenvolver. A secção 2.6 descreve de forma detalhada os produtos selecionados, o GridGain suportado por Apache Ignite e o Pivotal Gemfire suportado por Apache Geode.

A secção 2.7 refere a ferramenta de testes Yardstick, que será utilizada na criação do ambiente de testes aos protótipos de *software* IMDG implementados. Por fim, a secção 2.8 refere um trabalho relacionado com o desenvolvimento do estudo e a secção 2.9 uma breve síntese acerca do capítulo.

2.1. Big Data

Estamos em plena era da informação, onde o volume de dados produzidos diariamente em 2015 representa cerca de 2.2 milhões de *terabytes*, segundo a Gartner, citado por (Guerreiro 2015). A maior parte consiste na informação produzida pelas pessoas, como *emails*, vídeos, músicas, fotografias, publicações nas redes sociais, conversas de call centers, chamadas telefónicas, dados de GPS e cliques em web sites. A Figura 2 ilustra um infográfico com os números do que ocorre mundialmente na Internet a cada minuto, para uma melhor perceção do significado deste crescimento (Intel 2013).

Figura 2 – Um minuto na Internet, retirado de (Intel 2013) .

Presentemente, os sensores digitais ou *smart grids*, usados em equipamentos industriais, automóveis, habitações e medidores elétricos, contribuem hoje e cada vez mais num futuro bastante próximo para o crescimento substancial do volume de dados. Estes podem medir e comunicar a localização, o movimento, a vibração, a temperatura e a humidade (D.Schneider 2012). A inserção de micro transmissores nestes equipamentos permitem transmitir informações significativas, que são utilizados para fazer a monitorização preventiva em redes inteligentes, na conexão de diversos dispositivos à Internet, que gerando informação útil em tempo real tem o grande benefício de melhorar a qualidade de serviço (QoS) ao cliente ao nível do controlo, previsão e monitorização de custos e possíveis falhas.

Os especialistas preveem que haverá uma explosão de dados gerados pela aplicação de micro transmissores em máquinas, o que poderá representar um crescimento duas vezes maior que o crescimento de dados gerados por pessoas. Até ao final de 2015, a previsão era que cerca de seis milhares de milhões de objetos no mundo estivessem conectados à internet, o que é conhecido como IoT (Internet das Coisas) (Plat & Kekligian 2009).

O conceito *Big Data* é utilizado para descrever o crescimento, a disponibilidade e o uso exponencial de dados que requerem ser armazenados, processados, analisados e transformados em conhecimento para uma melhor perceção na tomada de decisão e movimentos estratégicos por parte das organizações ao nível de negócios. Também é usada para se referir a conjuntos de dados enormes e complexos com diferentes formas de organização, incluindo dados estruturados, semiestruturados e não estruturados (Grolinger et al. 2013). Este conceito destaca a capacidade de armazenar grandes volumes de dados que se pretendem analisar em quase tempo real, criando valor para as próprias organizações. As principais características do *Big Data* são (Canaltech 2014) : volume; variedade; velocidade; veracidade e valor, como se observa na Figura 3.

- Volume – fica a dever-se à quantidade de dados gerados todos os dias. Por exemplo, o Facebook produz num dia cerca de 10 milhares de milhões de mensagens, 4,5 milhares de milhões de gostos e 350 milhões de fotos compartilhadas. Os dados estão a deixar a escala do *terabyte* e atingem hoje a escala do zettabyte. O *Big Data* serve exatamente para lidar com esse volume de dados, guardando-os em diferentes lugares e juntando-os através de *software* específico.

- Variedade - no passado, a maior parte da informação era estruturada e podia ser colocada em bases de dados com tabelas e relações. Hoje, isso é um pouco difícil porque o grande volume de dados existentes são não-estruturados. A Gartner prevê que existam cerca de 10% de dados estruturados e 90% de dados não-estruturados (Guerreiro 2015). O *Big Data* permite que esta percentagem de dados não-estruturados, derivados de fotos, mensagens, vídeos, músicas, entre outros, possam ser administrados juntamente com os dados tradicionais estruturados.

- Velocidade – pelo motivo dos dados serem criados, transformados e processados em quase tempo real. Basta pensar na quantidade de eventos e mensagens que se espalham em instantes de segundos pelas redes sociais, as transações de cartão de crédito verificadas a cada instante ou até os milissegundos necessários para o cálculo do valor de compra e venda de ações. O Big Data serve para analisar os dados no instante em que são criados, em tempo-real, sem ter que proceder ao armazenamento em bases de dados.

- Veracidade – dado que é importante no mundo da informação verificar que esta seja verdadeira. Com o *Big Data* é impossível ter o controlo total de cada *hashtag* do Twitter ou notícia falsa na internet, mas com análises estatísticas de grandes quantidades de dados é possível compensar as informações incorretas.

- Valor - é o que torna o *Big Data* relevante, sendo este importante para gerar valor útil à organização. As organizações modernas têm entrado no negócio do *Big Data* por isto mesmo, para acrescentar valor a elas próprias, como estratégia ao nível de negócios, tentando perceber os comportamentos das pessoas e mercados.

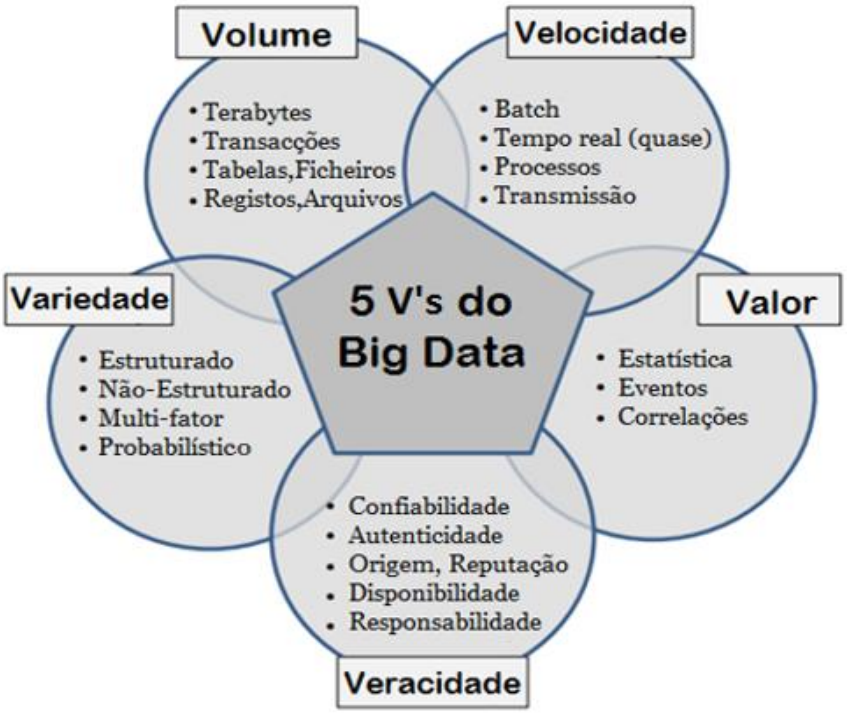


Figura 3 - Os 5 V's. Big Data, adaptado de (Demchenko et al. 2013) .

No seguimento do crescimento do termo *Big Data*, prevê-se que a informação gerada proveniente de diversos tipos de fontes de dados represente, em 2020, cerca de 35.2 zettabytes, ou seja, 44 vezes mais do que os 0.8 zettabytes produzidos em 2009, como podemos observar na Figura 4 (Borges 2013) .

Figura 4 - Crescimento da Informação na última década, retirado de (Borges 2013).

Perante este cenário, é importante que as empresas, governos e outras instituições, consigam lidar com este crescimento de dados e tirar o melhor proveito dos mesmos ao nível de BI (Inteligência Empresarial) com estruturação e criação de relatórios significativos e *Data Science* através de análises preditivas para a previsão de acontecimentos. Assim será possível tirar grande vantagem da existência do *Big Data*, no que toca à redução de custos e tempo, desenvolvimento de novos produtos e tomada de decisão inteligente.

O *Big Data* oferece esta potencialidade, a de revolucionar a maneira de fazer negócio, fornecendo novas perspetivas sobre as empresas, incluindo a forma de interação entre a empresa e o cliente, a forma de entregar produtos e serviços para o mercado, a posição da organização frente aos seus concorrentes e estratégias a adotar para aumentar a rentabilidade, o que pode ser entregue em tempo quase real através de processamento e análise de informação.

Perante os desafios das RDBMS tradicionais e a forma como lidamos com o *Big Data*, um número de soluções especializadas tem emergido nos últimos anos, numa tentativa de responder a estas preocupações. Novas abordagens, como NewSQL, NoSQL, IMDBMS e as soluções de armazenamento distribuído de dados em grid na memória RAM como as IMDGs, têm-se apresentado como alternativas viáveis para o processamento de dados que lidam com o crescente volume de dados, fornecendo alto desempenho e a escalabilidade necessária.

2.2. NewSQL

As RDBMS tradicionais vão de encontro a três princípios fundamentais para garantir a segurança da informação: confidencialidade, disponibilidade e integridade (Lauriano 2005). A Confidencialidade certifica que a informação armazenada apenas pode ser disponibilizada a pessoas que sejam explicitamente autorizadas, pelo processo de identificação e autenticação das partes envolvidas. A Disponibilidade garante que a informação armazenada é disponibilizada no momento em que esta for necessária. A Integridade assegura o retorno da informação armazenada na sua forma original, protegendo as informações ou dados contra modificações acidentais ou não-autorizadas. Para além disso, as RDBMS asseguram propriedades ACID, garantindo que as operações de base de dados são processadas de forma confiável. As RDBMS tem tido uma posição dominante ao longo das últimas décadas nas operações de armazenamento e processamento de dados, no que se refere a inserir, selecionar, atualizar e eliminar (CRUD), mas hoje enfrentam grandes problemas de desempenho nestas operações (Ji et al. 2012).

O NewSQL oferece um conjunto de novas bases de dados relacionais que proporcionam um melhor desempenho, escalabilidade horizontal e tolerância a falhas em relação às RDBMS tradicionais, mantendo no entanto a utilização da linguagem SQL e suas propriedades ACID características. Devido à crescente produção de dados gerados diariamente de diversas fontes, este tipo de bases de dados adequam-se à resolução de problemas tradicionais no armazenamento e processamento de informação em que é exigido um maior nível de requisitos de desempenho e escalabilidade, bem como uma maior complexidade na implementação de sistemas (Abigail & Ribeiro 2013). A sua utilização é apropriada em sistemas em que a estrutura de dados é conhecida antecipadamente e não muda, como é o caso do modelo relacional, e em aplicações que requerem a manipulação de objetos ou tem requisitos fortes de consistência, como é o exemplo de aplicações no mercado financeiro. Por exemplo, numa ação de transferência de dinheiro entre duas contas distintas na operação de atualização logo após efetuada a transação (João 2015).

A base de dados NewSQL representa uma grande vantagem no que se refere ao armazenamento de dados estruturados, pelo facto de ser compatível com a maioria das tecnologias DBMS, ao nível de propriedades e linguagem SQL tradicional, e ainda assim oferecer

uma maior capacidade para a implementação de sistemas distribuídos complexos de alto desempenho. Alguns exemplos de NewSQL DBMS são: VoltDB; ClustrixDB; Schooner MySQL; ScaleBase (Dubray 2011).

2.3. NoSQL

Para lidar com o crescente volume e variedade de dados semiestruturados e não-estruturados, derivados de documentos, textos, imagens, vídeos, entre outros, emergiu uma nova categoria de bases de dados não-relacionais, o NoSQL (Barreiro 2014) . Esta apresenta características como: modelos de dados não-relacionais simples e flexíveis; tolerância a falhas, capacidade de escalar horizontalmente com recurso a máquinas num ambiente distribuído; alta disponibilidade (Abigail & Ribeiro 2013). Não tendo suporte para transações ACID, o NoSQL oferece propriedades BASE. O atributo "Basicamente Disponível" significa que a base de dados está disponível o tempo todo sempre que for requisitado mesmo que algumas partes não estejam disponíveis. O "Estado Leve" significa que a base de dados não precisa de ser sempre consistente, podendo tolerar a inconsistência durante um período de tempo. O "Eventualmente Consistente" significa que após um período de tempo, a base de dados chega a um estado consistente (Roe 2012).

A base de dados NoSQL está principalmente direcionada para cenários distribuídos e por isso devemos ter em consideração o Teorema CAP, ilustrado na Figura 5 (Roe 2012) (Souza 2013). O princípio central do teorema afirma que é impossível que um sistema distribuído de dados consiga cumprir os três requisitos: Consistência, Disponibilidade e Tolerância ao Particionamento, em simultâneo.

O primeiro refere-se ao facto de um sistema estar consistente, após a execução de uma operação. Por exemplo, um sistema distribuído é considerado consistente se após escrever um registo, este fica disponível a ser utilizado imediatamente. O segundo refere-se à implementação de um sistema de modo a que seja assegurado que esse permanece ativo durante um determinado período de tempo. O terceiro refere-se ao facto de um determinado sistema continuar a funcionar, mesmo sob condições de perda de dados ou falha do sistema, garantindo que uma possível falha de um nó não conduza ao colapso do sistema no seu todo. Tendo em consideração que o teorema

não se aplica apenas a base de dados NoSQL, é importante refletir e reconhecer os requisitos que queremos ver cumpridos na implementação de um sistema distribuído altamente disponível e escalável.

Figura 5 - Teorema CAP, retirado de (Sousa 2010).

As bases de dados NoSQL podem ser classificadas nos quatro seguintes grupos (Kumar 2014) (Pronschinske 2010):

- Pares Chave / Valor – utiliza uma tabela de dispersão com uma chave única e um ponteiro para um valor em particular. Estes caracterizam-se por ser extremamente simples e facilmente escaláveis. Alguns exemplos de tecnologias DBMS NoSQL existentes para este grupo são o MemCacheDB, Redis, SimpleDB e HBase.

- Orientado a Documentos - são semelhantes a uma base de dados de pares Chave / Valor, mas a única diferença é que os valores armazenados (referido como "documentos") fornecem alguma estrutura e codificação na gestão de dados (como XML, JSON, etc). Alguns exemplos de tecnologias DBMS NoSQL existentes para este grupo são o CouchDB, MongoDB, BaseX e RavenDB.

- Orientado a Coluna – procede ao armazenamento de dados em células, que são agrupados em colunas de dados, em vez de linhas de dados. As operações de leitura e

escrita também são feitas utilizando colunas em vez de linhas. Alguns exemplos de tecnologias DBMS NoSQL existentes para este grupo são o Cassandra e Hypertable.

- Baseada em Grafos - utiliza grafos para o armazenamento de dados não-estruturados, representando-os na forma de esquema como grafos dirigidos, ou como estruturas que generalizem a noção de grafos. Alguns exemplos de tecnologias DBMS NoSQL existentes para este grupo são o Neo4j, OrientDB e InfiniteGraph.

2.4. Sistemas de Gestão de Bases de Dados em Memória (IMDBMS)

Os RDBMS tradicionais tem tido uma posição dominante ao longo das últimas décadas. No entanto, a sua escalabilidade e desempenho limitado ao nível de processamento de grandes volumes de transações *online*, eventos complexos e de *Big Data* tem influenciado o crescimento dos IMDBMS (Ji et al. 2012).

Para além de manterem as qualidades conhecidas de uma base de dados relacional tradicional, os IMDBMS tem a particularidade de fazer o armazenamento e manipulação de dados utilizando recursos de memória RAM ao invés de discos rígidos (Vamsi & Venkatesh 2012). Um fator que contribuiu para a mudança de paradigma foi o facto do preço da RAM ter descido substancialmente e a capacidade da memória ter aumentado drasticamente, retirando grande vantagem na utilização desta abordagem no que se refere à rapidez nos tempos de acesso aos dados.

Nos tempos que correm as organizações devem estar preparadas para suportar grandes cargas de trabalho *online* e executar consultas complexas. Os sistemas OLAP (Processamento Analítico Online) contribuem significativamente para este facto, especialmente em aplicações de BI, onde é necessário satisfazer as exigências das organizações no que toca à análise dos seus dados, a fim de ajudá-los a tomar melhores decisões ao nível de negócios. Estas organizações movem os seus dados para um armazém de dados (Data Warehouse) e, em seguida, preparam relatórios pré-definidos, a fim de ganhar novas capacidades analíticas, sendo essencial que as bases de dados estejam preparadas para trabalhar sobre elevado desempenho e tempo de resposta (Gayberi et al. 2013).

As IMDBMS permitem um armazenamento mais eficiente e um processamento de dados muito mais rápido, com baixa latência e sobrecarga de disco rígido devido a algoritmos de otimização simples com poucas instruções de CPU executadas.

Estes factos estão comprovados por empresas de grande escala e tecnologicamente avançadas, tais como, a Altibase ou Oracle, que comprovam melhorias significativas ao nível de performance e disponibilidade, com tempos de resposta acima da média, quando comparamos o acesso e manipulação de dados em base de dados residentes na memória e no disco rígido.

A Altibase fez um estudo onde é possível comparar a performance da sua ferramenta de base de dados em memória com outras que utilizam bases de dados tradicionais em disco em operações de "INSERT", "SELECT", "DELETE" e "UPDATE", medindo o acesso e processamento de transações por segundo. Os resultados mostram na Figura 6, que a performance em memória para o tipo de operações mencionadas acima é bastante mais rápida do que em disco, sendo possível que ações de "SELECT" em memória sejam entre duas e cinco vezes mais rápidas do que em disco (Altibase 2014).

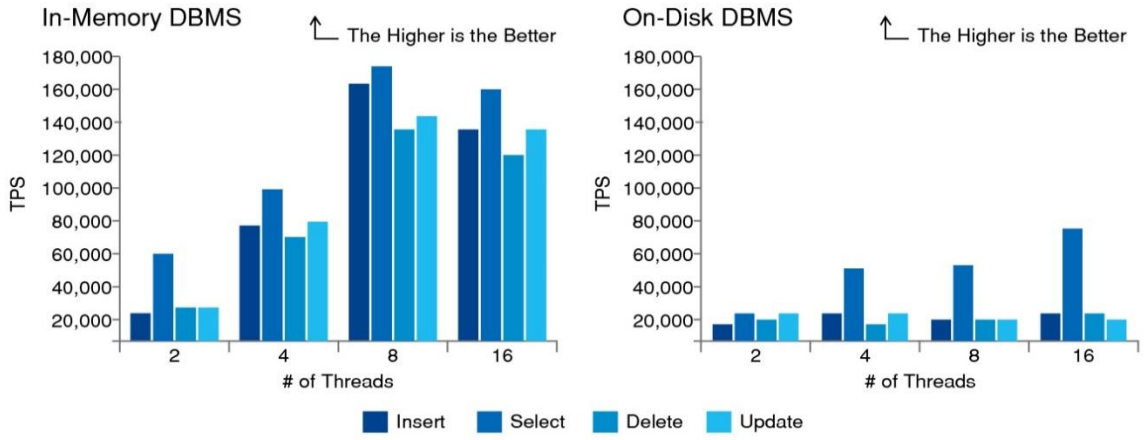


Figura 6 - Comparação de performance IMDBMS e On-Disk DBMS, retirado de (Altibase 2014).

Atualmente a oferta é enorme e existem diversas tecnologias *open source* e comerciais para trabalhar com bases de dados em memória, tais como, a Altibase, a Sap Hana, extremeDB, Oracle Database 12 c, Redis, VoltDB e Unicom SolidDB. Os IMDBMS são recomendados principalmente em aplicações que exigem a visualização e manipulação de dados em tempo real e que requerem um alto desempenho como é o caso das áreas das telecomunicações, finanças, defesa e inteligência.

2.5. *Grid* de Dados em Memória (IMDG)

Nos últimos anos, muito trabalho tem sido feito para ultrapassar os problemas de consistência e persistência de dados, bem como aumentar a rapidez e capacidade no processamento de grandes conjuntos de dados em sistemas de alta velocidade. Esforços como o Impala, Storm, Spark e mais recentemente HAWQ tem caminhado neste sentido através de abordagens diferentes. O Impala e HAWQ concentram-se principalmente em oferecer mecanismos de consulta rápida de dados sobre o Hadoop. O Storm faz o processamento de fluxos de dados em eventos, e Spark é uma infraestrutura em memória que permite de forma muito significativa acelerar o desempenho em processamentos paralelos, ao estilo Map Reduce (Williams et al. 2014). Embora cada um destes sistemas sejam um passo muito positivo no sentido de encontrar uma solução híbrida, nenhum deles foi construído com o objetivo de ser uma plataforma completa, combinando alta velocidade com alto volume de dados.

Posto isto, as tecnologias IMDG aparecem como uma alternativa promissora para colmatar o fosso dos métodos e sistemas existentes, fazendo a junção de diversas funcionalidades numa plataforma única. A Gartner prevê que o mercado das IMDGs chegará perto de um milhar de milhão de dólares em 2016, devido à sua capacidade de armazenamento, execução de consultas avançadas e realização de transações complexas em grandes volumes de dados de forma rápida e escalável sem comprometer a disponibilidade, consistência e integridade dos dados (Rivera & Meulen 2013).

Esta distingue-se como uma plataforma que armazena e processa dados num sistema distribuído com escalabilidade de máquinas, formando uma estrutura de dados residentes numa *grid* inteiramente em memória, atingindo assim níveis elevados de armazenamento e processamento de volumes de dados, para além de oferecer alta performance, disponibilidade e escalabilidade.

Para que isto seja possível é importante que as IMDGs trabalhem paralelamente com o conceito de IMCG (Computação de Grid em Memória). Enquanto as IMDGs se focam essencialmente no armazenamento distribuído e gerenciamento de grandes conjuntos de dados pelo particionamento dos mesmos na *grid* em memória entre várias máquinas disponíveis no sistema distribuído, as IMCGs concentram-se essencialmente em algoritmos eficientes para o

processamento de dados nos diversos nós do *cluster*. Em suma, IMDG representa o armazenamento e gestão de dados em memória, e IMCG representa o processamento e computação dos mesmos (Ivanov 2012).

O fator diferenciador deste tipo de plataformas é a utilização da memória principal para o armazenamento ao invés do disco rígido, suprindo as limitações de desempenho, embora se possa argumentar que os discos SSD (Unidade de Estado Sólido), a evolução dos discos rígidos tradicionais, sejam uma alternativa viável, devido ao seu desempenho consideravelmente mais rápido. Contudo, a latência[1] deste tipo de discos ainda está numa ordem de magnitude mais lenta, relativamente às velocidades médias da memória RAM (Memória de Acesso Aleatório) (Williams et al. 2014), chegando a memória principal a ser 800 vezes mais rápido do que um HDD e até 40 vezes mais rápido do que um disco SSD ao nível de processamento de dados (Prins 2013).

A utilização da memória RAM com certeza não cairá em desuso porque já se fala na evolução da RAM para a MRAM, tecnologia não volátil que utiliza cargas magnéticas para o armazenamento e acesso aos dados, atingindo desempenhos superiores à RAM. Por ser uma tecnologia não volátil, ao contrário da RAM, a energia poderá ser desligada e os dados serão mantidos, isto porque o armazenamento é feito através de campos magnéticos, ao invés de circuitos elétricos (Technologies 2015). Pode-se assim antecipar que as tecnologias de armazenamento e processamento de dados em memória terão um futuro duradouro.

Isto será importante sobretudo em operações ou aplicações que fornecem informação em quase tempo real e que necessitam de respostas instantâneas, tais como: carregamentos *online*; serviços baseados em localização; dados e eventos de mercado; análises de dados para *dashboards* interativas; vinculação de anúncios; entre muitos outros.

2.5.1. Características

As tecnologias IMDG definem-se como um grid para armazenamento de dados distribuídos em *clusters* de máquinas inteiramente na memória sobre uma estrutura de dados orientada a objetos. Por forma a complementar o ADN deste tipo de tecnologias procedemos de seguida à

1 Período de espera ou demora para obter o resultado esperado.

formulação em detalhe das características que a tornam vantajosa em relação a outras tecnologias existentes (Prins 2013) (Williams et al. 2014).

- Escalabilidade elástica e linear - Suporta o crescimento uniforme recorrendo à escalabilidade horizontal.
- Grid de Dados dinâmico - Adiciona ou remove servidores em tempo de execução sem interrupção para aumentar a quantidade de memória disponível.
- Cache distribuída em memória - Utiliza APIs específicas para interagir com caches na memória.
- Ferramentas de gestão e monitorização - Acesso em tempo real à gestão e análise dos dados na memória.
- Linguagem de Suporte - Ambientes de desenvolvimento em Java e Scala.
- Resiliência[2] e FailOver - Alto desempenho (grandes cargas de processamento) e rapidez com tolerância a falhas na recuperação de falhas (servidores em baixo).
- Baixa latência no armazenamento e acesso a dados - O período de espera pela obtenção de resultados é baixo.
- Estrutura de dados Chave/Valor - Armazenamento de dados simples ou objetos complexos em pares Chave/Valor.
- Políticas de Substituição e Expiração - Define estratégias de remoção de entradas aquando a cache fica totalmente preenchida ou após um período de tempo.
- Recurso a transações "Fully" ACID - (Atomicidade, Consistência, Isolamento, Durabilidade), que caracteriza uma transação.
- Consultas avançadas - Incorpora ferramentas de pesquisa (linguagem SQL e outras) de dados no sistema distribuído em memória.
- Suporte para Streaming - Processamento distribuído de grandes fluxos de dados de streaming em tempo real.

2 Tolerante a falhas, capacidade do sistema distribuído operar na presença de falhas.

- Suporte para Hadoop – Oferece recursos de HDFS (Sistema de Arquivos Distribuídos Hadoop) para o armazenamento e acesso a dados a partir de sistemas de arquivos, e Map Reduce para o processamento de grandes quantidades de dados em paralelo num *cluster* de máquinas.

Para além das características mencionadas, as tecnologias IMDG fornecem diversas topologias de armazenamento de dados no sistema distribuído em memória, tais como: Local, Particionada e Replicada (Matyashovsky 2013).

- Local: O armazenamento de dados é feito no nó ou servidor local de forma não distribuída pelos restantes nós ou servidores do *cluster*. A topologia local tem a vantagem de ser simples pelo facto de não existir distribuição de dados por outros nós do *cluster*, mas por outro lado tem a desvantagem de não ser escalável e tolerante a falhas.
- Particionada: O armazenamento de dados é feito com a divisão do conjunto de dados, em partições, que por sua vez são vez são igualmente distribuídas em todos os nós ou servidores do *cluster*. A topologia particionada tem a vantagem de ser escalável, tolerante a falhas e oferecer alta performance (menos latência) nas escritas pelo facto de ser apenas necessário escrever os dados na partição do nó primário e de *backup*, mas por outro lado tem a desvantagem de baixa performance (mais latência) nas leituras, pois os dados não se encontram em todos os nós, o que requer uma procura desses dados nos nós.
- Replicada: O armazenamento de dados é feito com a replicação dos mesmos por todos os nós ou servidores do *cluster*. A topologia replicada tem a vantagem de ser escalável, tolerante a falhas e oferecer alta performance (menos latência) nas leituras pelo facto dos dados se encontrarem em todos os nós do *cluster*, mas por outro lado tem a desvantagem de ter uma baixa performance (mais latência) nas escritas, pois é necessário escrever os dados em todos os nós do *cluster*, o que requerer carga de rede e memória adicionais.

Algo importante a referir nas tecnologias IMDG, e que estão diretamente relacionadas com as topologias de armazenamento, são os processos de comunicação em operações de leitura e escrita dos dados nos nós do *cluster*, que podem ser feitas de forma síncrona ou assíncrona (Schuncke 2013).

- Síncrona (*Write/Read-Through*): Neste tipo de comunicação de dados os dispositivos emissor e recetor devem estar sincronizados antes da comunicação iniciar e durante a transmissão dos dados. Como ilustra a Figura 7, quando o bloco de dados (Requisição I) é enviado pelo emissor, este é bloqueado até que o recetor o receba, e só depois é que enviamos outro bloco de dados (Requisição II).

Figura 7 - Requisição Síncrona.

- Assíncrona (*Write/Read-Behind*): Neste tipo de comunicação de dados, um bit especial é inserido no início e no fim do bloco a transmitir, permitindo ao recetor entender o que realmente foi transmitido. Como ilustra a Figura 8, o bloco de dados (Requisição I) é enviado pelo emissor com o bit especial que informa o recetor onde começa e acaba o bloco. Isto permite que os restantes blocos de dados (Requisição II e III) possam ser transmitidos em qualquer ordem, pois cabe ao recetor interpretar a informação e colocá-los no lugar certo.

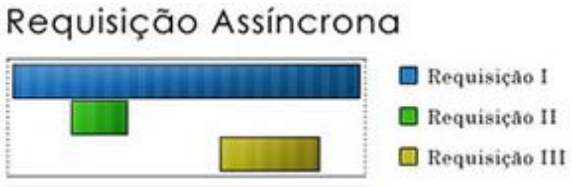


Figura 8 - Requisição Assíncrona.

2.5.2. Capacidade e Confiabilidade

Para utilizar a memória principal como uma área de armazenamento é essencial superar dois pontos fracos. O primeiro é a capacidade limitada, que envolve o armazenamento de dados excedendo a capacidade máxima de memória disponível pelos servidores. O segundo é a confiabilidade, que envolve a perda de dados no caso de falha de um servidor.

Uma IMDG ultrapassa o problema da capacidade com a utilização de uma arquitetura distribuída assegurada pela escalabilidade horizontal, e o problema da confiabilidade através de

topologias de armazenamento replicado e/ou configuração de propriedades de *backup* dos dados nos servidores do cluster, que única e exclusivamente tenham a função de salvaguardar toda a informação em caso de falha do sistema (Prins 2013).

2.5.3. Estrutura de Dados

O modelo de dados utilizado nas tecnologias IMDG é baseado em pares de Chave/Valor. A chave é utilizada para armazenar e recuperar os valores numa lista e pode ser comparada com o índice de chave primária de uma tabela numa base de dados tradicional. As IMDGs estão intimamente ligadas a ambientes de desenvolvimento em Java. Por isto, os valores dos pares Chave/Valor podem ser qualquer tipo suportado pelo ambiente Java, que vão desde tipos de dados simples (int, string), a objetos complexos. A Tabela 1 mostra um exemplo de como podemos formar pares Chave/Valor com um *"id"* que representa a chave e um valor formado por um objeto *" (Organização: {id,fundador,empresa}) "*.

Chave (id)	Valor (Organização: {id,fundador,empresa})
"Cod101"	{"101, Abe Kleinfeld, GridGain"}
"Cod102"	{"102, Talip Ozturk, Hazelcast"}
"Cod103"	{"103, John Doe, PivotalGemFire"}

Tabela 1 - Exemplo de pares Chave/Valor.

Assim sendo, será possível armazenar objetos como um valor, não sendo necessária qualquer tradução de objetos num modelo de dados relacional. O que aparentemente poderia ser um obstáculo para a criação do nosso modelo de dados, efetivamente não é nenhuma limitação em muitos casos de utilização.

A Figura 9 mostra uma arquitetura IMDG com um conjunto de chaves {k1,k2,k3} e um conjunto de valores {V1,V2,V3} onde cada chave pertence a um nó ou servidor diferente. Neste caso podemos fazer a leitura de dados a partir de ficheiros em qualquer formato (JSON, CSV, XML, TXT) ou em bases de dados, e armazenar na arquitetura distribuída em memória os pares de Chave/Valor. Opcionalmente é possível recorrer a uma base de dados externa para escrever os pares Chave-Valor, garantindo a persistência dos dados. Se assim for, será necessário recorrer a bases de dados NoSQL (Hall 2013), como verificámos na secção 2.3, para o armazenamento deste tipo de estruturas.

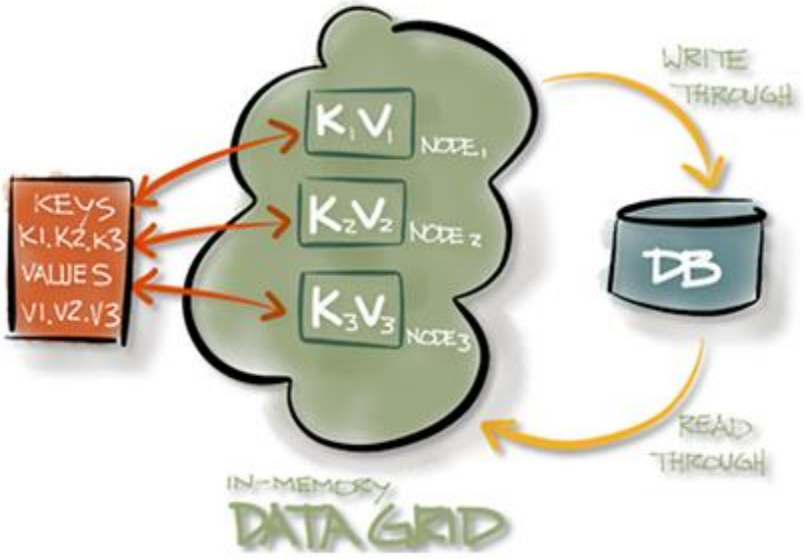


Figura 9 - Arquitetura IMDG, retirado de (Setrakyan 2012).

2.5.4. Performance, Escalabilidade e Alta Disponibilidade

As grandes vantagens da utilização de tecnologias IMDG em sistemas de grande escala é que estas conseguem atingir os objetivos ao nível de performance, escalabilidade e disponibilidade no processamento e armazenamento de dados na memória.

A Performance ao nível de sistemas refere-se a um conjunto de características ou capacidades de comportamento e rendimento quando comparados com metas, requisitos ou expectativas previamente definidas. Na área da computação de alta performance (HPC), esta refere-se à utilização de supercomputadores baseados em *clusters* de vários computadores ou servidores de tarefas que requerem grandes recursos de computação. Recentemente, este termo tem vindo a ser aplicado na área de negócios que lidam com armazéns de dados (Data Warehouse) e processamento complexo de transações (Wikipédia 2013) .

A Escalabilidade é a capacidade que uma organização ou sistema possui para o crescimento atendendo às necessidades sem perder as qualidades que lhe acrescentam valor, ou seja, é estar preparado para o crescimento consoante o aumento da carga de trabalho. Nos dias que correm, uma solução escalável é uma condição de sucesso, o que permite que estas sejam rapidamente absorvidas pelo mercado (Bondi 2000). Por exemplo, um sistema de processamento de transações ou um sistema de gestão de base de dados é escalável quando é possível atualizá-lo

para aumentar a capacidade de transações, mediante a adição de novos processadores, mecanismos e dispositivos de modo transparente sem ser necessário desligar o sistema. Existem duas formas de adicionar recursos a um sistema escalável (Informant 2014):

- Escalabilidade Horizontal: consiste em adicionar mais capacidade ao sistema distribuído inserindo uma nova máquina num *cluster* de máquinas.
  - **Vantagens:** *Hardware* relativamente barato; escalabilidade virtualmente ilimitada.
  - **Desvantagens:** Aplicável apenas quando o sistema é projetado para ser horizontalmente escalável; requer administração adicional.

- Escalabilidade Vertical: consiste em fazer um *upgrade* num servidor já existente na rede, ou seja, substituir, repor ou agregar mais recursos, como memória ou discos rígidos, por exemplo.
  - **Vantagens:** Processo relativamente simples; transparente para a arquitetura e desenvolvimento de sistemas.
  - **Desvantagens:** *Hardware* relativamente caro; limitado ao nível de escalabilidade.

As tecnologias IMDG tornam-se altamente escaláveis pelo facto de utilizarem a escalabilidade horizontal, recorrendo à virtualização e capacidade de monitorização constante de máquinas que permitam controlar pedidos, distribuir carga de processamento e alocar recursos de acordo com as necessidades em tempo real.

Por último, um sistema de Alta Disponibilidade é um sistema tolerante a falhas de *hardware*, software ou energia, cujo objetivo é manter os serviços disponibilizados o máximo de tempo possível (Wikipédia 2013b). As IMDGs recorrem a topologias de armazenamento particionado ou replicado com propriedades de *backup* configuradas e redundância de servidores de cache para garantir a persistência de dados em caso de interrupções e falhas dos mesmos, mantendo a disponibilidade do sistema ao mais alto nível.

2.5.5. Comparação de Tecnologias IMDG

Atualmente, a oferta é grande e existem inúmeros produtos IMDG com versões *open source* e comerciais para o armazenamento e processamento de dados, tais como: Hazelcast;

Infinispan; TayzGrid; Apache Ignite/GridGain; Apache Geode/Pivotal Gemfire; XAP Giga Spaces; Terracota; Oracle Coherence.

A Tabela 2 disponibiliza uma síntese de cada um destes produtos baseada nos principais recursos das tecnologias IMDG. Esta foi realizada recorrendo à documentação pública *online* e a uma plataforma oferecida pelo produto da GridGain que permite fazer a comparação entre produtos. Assim foi possível fazer as seguintes comparações: GridGain vs. Hazelcast (Systems 2015); GridGain vs. Infinispan (Gridgain Systems 2014); GridGain vs. TayzGrid (TayzGrid 2015); GridGain vs. Pivotal Gemfire (GridGain Systems 2015); GridGain vs. XAP GigaSpaces (GridGain Systems 2014); GridGain vs. Terracota (Gridgain Systems 2014); GridGain vs. Oracle Coherence (GridGain Systems 2014).

Perante a informação disponível procedeu-se à seleção de dois produtos para o desenvolvimento do trabalho experimental à funcionalidade de *caching* de relatórios na plataforma NPM, com base nos seguintes requisitos: disponibilidade de versões *open source*, produtos líderes de mercado e semelhança na complementaridade de recursos. Desta forma, foram selecionados o Apache Ignite/GridGain e Apache Geode/Pivotal GemFire que serão detalhados nas secções seguintes.

Produto	Hazelcast	Infinispan	TayzGrid	Apache Ignite/ GridGain	Apache Geode/ Pivotal GemFire	Giga Spaces (XAP)	Terracota	Oracle Coherence
Versão								
Open Source	●	●	●	●	●	●	●	●
Proprietária	● Enterprise	●	● Enterprise & Professional	● Enterprise	●	● Premium	● BigMemory Max&Go	● Coherence 12cR2 12.2.1
Características IMDG								
Suporte JCache	●	●	●	●	●	● JavaMap	● Ehcache	●
Transações ACID	●	●	●	●	●	●	●	●
Elasticidade (add/remove nodes)	●	●	●	●	●	●	●	●
SQL Queries	● Distributed SQL-Like Query	● Query DSL	● Object Query Language - OQL	● ANSI SQL	● Object Query Language - OQL	● JPQL (JPA Query Language)	● Query Manager	● (custom coherence query language)
SQL Indexes	●	●	●	●	●	●	●	●
Distribuições SQL (Join & Aggregation)	●	●	?	●	●	●	●	●
Políticas de Expiração e Substituição	● LRU,LFU	● LRU	● LRU,LFU	● LRU,FIFO	● LRU	● LRU	● LRU,LFU,FIFO	● LRU,LFU
Memória On-Heap & Off-Heap	●	●	?	●	●	●	●	●
JDBC Driver	●	●	●	●	●	●	●	●

Ferramenta de Gestão e Monitorização								
GUI (graficamente)	●	●	●	●	●	●	●	●
Linha de Comandos	●	●	●	●	●	●	●	●
Topologias de Armazenamento								
Local	●	●	●	●	●	●	●	●
Particionada	●	●	●	●	●	●	●	●
Replicada	●	●	●	●	●	●	● Limitado	●
Suporte In-Memory Streaming								
Branching Pipelines (Fluxos de Trabalho para Processamento Stream)	●	●	● Disco	●	●	● Disco	● Disco	●
Processamento de Eventos Complexos (CEP)	●	●	● Disco	●	●	● Disco	● Disco	●
SQL Queries sobre "sliding windows"	●	●	● Disco	●	●	● Disco	● Disco	●
Suporte In-Memory Hadoop								
HDFS	●	● Disco.	●	●	● Disco.	●	●	●
Map/Reduce	● Disco	● Disco	● Disco	●	● Disco	● Disco	● Disco	● Disco

Legenda: ? Sem Informação ● Disponível ● Limitado ● Indisponível

Tabela 2 - Comparação de Tecnologias IMDG.

2.6. Tecnologias IMDG Selecionadas

2.6.1. Apache Ignite / GridGain

A plataforma IMDG GridGain assenta no projeto Apache Ignite e permite fazer o armazenamento de grandes volumes de dados em memória recorrendo a *clusters* de máquinas distribuídas. Utiliza um modelo de dados baseado em pares Chave/Valor desenhado tendo em vista facilitar o escalonamento horizontal dos sistemas. Atualmente é vista como uma tecnologia de alto desempenho e escalabilidade com capacidade para processar grandes conjuntos de dados em memória e em tempo real, algo que pode não ser possível utilizando as tecnologias tradicionais de HDD.

O GridGain foi nomeado pela Gartner como uma das Tecnologias de Computação em 2014, concluindo na sua análise que "A capacidade de combinar tecnologias IMDG e IMDBMS com o processamento complexo de eventos numa única plataforma integrada na memória representa um passo em frente na agregação de tecnologias IMC em plataformas mais amplas e abrangentes" (Gideon 2014). Isto porque o GridGain oferece uma ampla arquitetura de recursos na memória, como podemos visualizar na Figura 10.

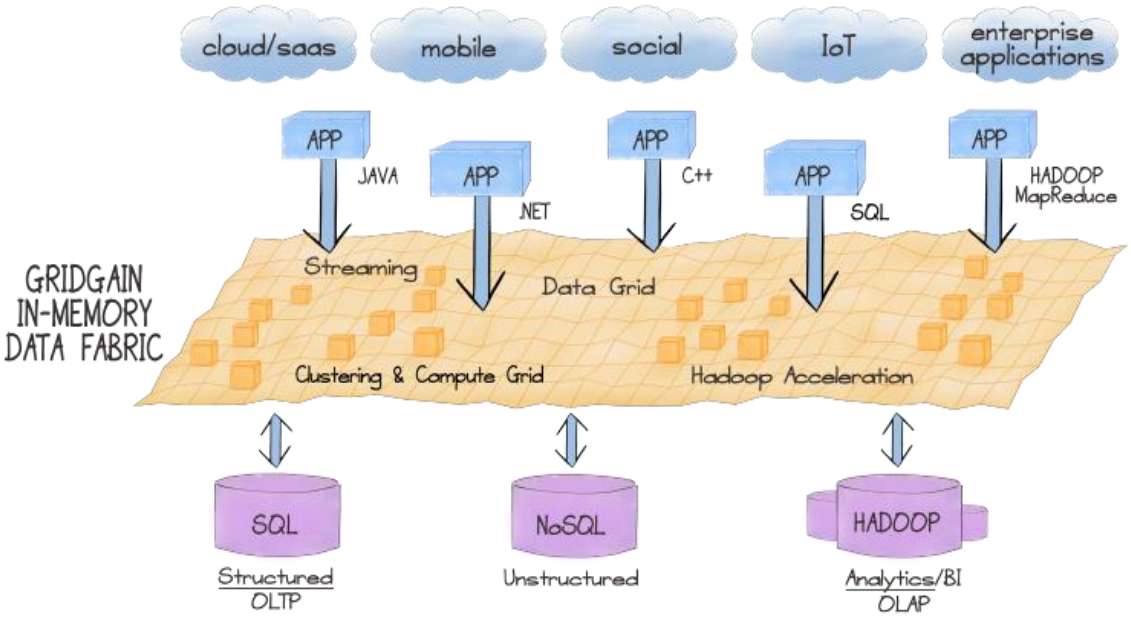


Figura 10 - Arquitetura GridGain IMDG Data Fabric, retirado de (Gutierrez 2014).

O produto tem dois tipos de versões, a GridGain Enterprise Edition ao nível comercial e a GridGain Community Edition ao nível *open source*, ambas suportadas pelo projeto Apache Ignite. A informação a seguir descrita refere-se às características do ecossistema do produto, retirado da documentação pública disponibilizada *online* [3].

A *cache* é distribuída em memória, On-Heap e/ou Off-Heap, modos que permitem fazer uma gestão estruturada dos dados, com o suporte da API JCache, que fornece uma interface simples e poderosa de armazenamento distribuído de dados com recurso a estruturas pares Chave/Valor, com possível configuração das políticas de substituição (LRU e FIFO) e de expiração de dados. Esta disponibiliza consultas avançadas na cache de dados, através do suporte a uma API de consulta com recursos para SQL Queries (ANSI 99), SQL Join e SQL Aggregation. O suporte para ferramentas de monitorização e gestão de dados na *grid* em memória permite ter um maior controlo de todos os nós do sistema distribuído. Se for importante trabalhar com sistemas OLTP de processamento de transações em tempo quase real numa organização, o GridGain oferece suporte para transações com características ACID.

Um dos recursos exclusivos é o suporte para sistema de arquivos distribuídos chamado de GridGain File System (GGFS). Este oferece funcionalidades semelhantes ao Hadoop Distributed File System (HDFS), mas com a particularidade de utilizar exclusivamente a memória. Para o processamento otimizado de grandes volumes de dados em paralelo, pela divisão de uma tarefa global em subtarefas independentes através de fases de mapeamento e redução do conjunto de dados, o GridGain disponibiliza o GridGain Map Reduce. Este recurso tem a particularidade de aceder ao conjunto de dados de entrada, produzir resultados intermédios e, em seguida, criar resultados inteiramente na memória, ao invés do Hadoop Map Reduce que o faz no disco.

Para processamento contínuo de fluxos de dados de forma escalável e tolerante a falhas, temos o GridGain Streaming que disponibiliza formas de transmissão, transformação e consulta aos dados a serem transmitidos, o que nos permite responder a perguntas como: "Quais são os 10 produtos mais populares das últimas 2 horas?".

[3] http://apacheignite.gridgain.org/

Algo bastante importante, é a forma como é feita a configuração da distribuição de dados no processo de armazenamento. O GridGain disponibiliza duas topologias de armazenamento fundamentais em *cache*, a Particionada e a Replicada que são ilustradas na Figura 11.

A topologia Particionada faz a distribuição do conjunto de dados pelas partições primárias dos nós do *cluster*, com a particularidade de armazenar em cada partição de *backup* o conjunto de dados de apenas um outro nó do cluster. A Figura 11 representa uma visão simples do modo de funcionamento da *cache* Particionada, em que a chave A é atribuída a um nó do *cluster* em execução na JVM1 (Máquina Virtual Java), B atribuída a um nó do *cluster* na JVM3, C atribuída a um nó do *cluster* em execução na JVM2 e D atribuída a um nó do *cluster* em execução na JVM4.

A topologia Replicada utiliza o modo de *cache* Particionada para a distribuição do conjunto de dados pelas partições primárias de cada nó do *cluster*, com a particularidade de armazenar na partição de *backup* o conjunto de dados de todos os outros nós do *cluster*. Na Figura 11, o nó em execução na JVM1 é um nó principal para a chave A, mas também armazena *backups* de todas as outras chaves (B,C,D). De notar que as vantagens e desvantagens na utilização das duas topologias mencionadas, ao nível de performance, estão referidas anteriormente na subsecção 2.5.1.

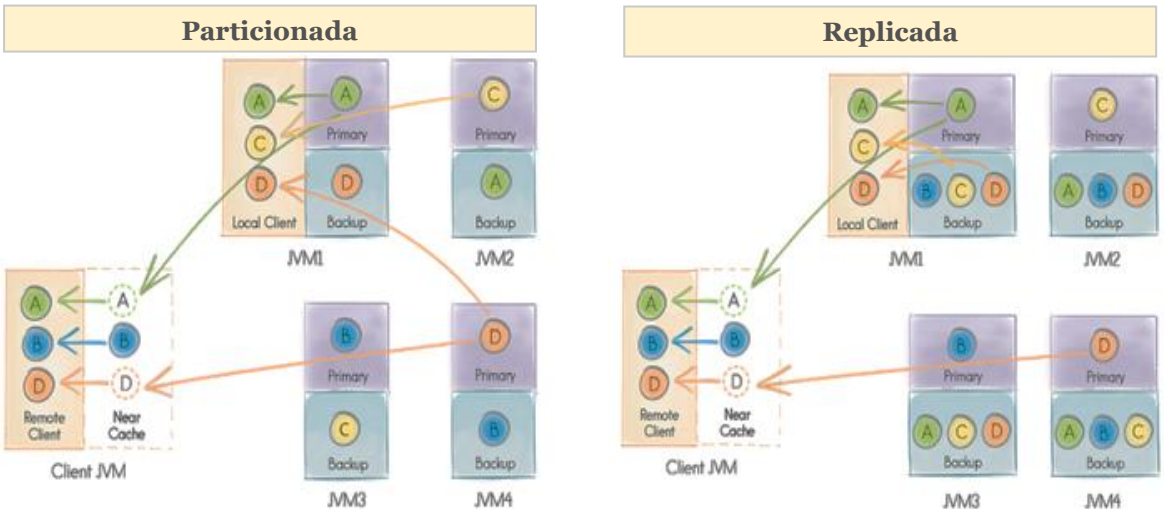


Figura 11 - Modo de Cache Particionada e Replicada no GridGain, retirado de (Apache Ignite 2015).

2.6.2. Apache Geode / Pivotal GemFire

O produto da Pivotal GemFire é uma plataforma IMDG que conta agora com as contribuições do projeto Apache Geode. Esta é baseada no armazenamento de dados de forma distribuída em memória pelos nós do *cluster* e é projetado para aplicações que exigem altos requisitos de escalabilidade e disponibilidade. Permite escalar elasticamente e horizontalmente, adicionando novos nós, aumentando a capacidade previsível e oferecendo uma variedade de recursos, como podemos observar na Figura 12.

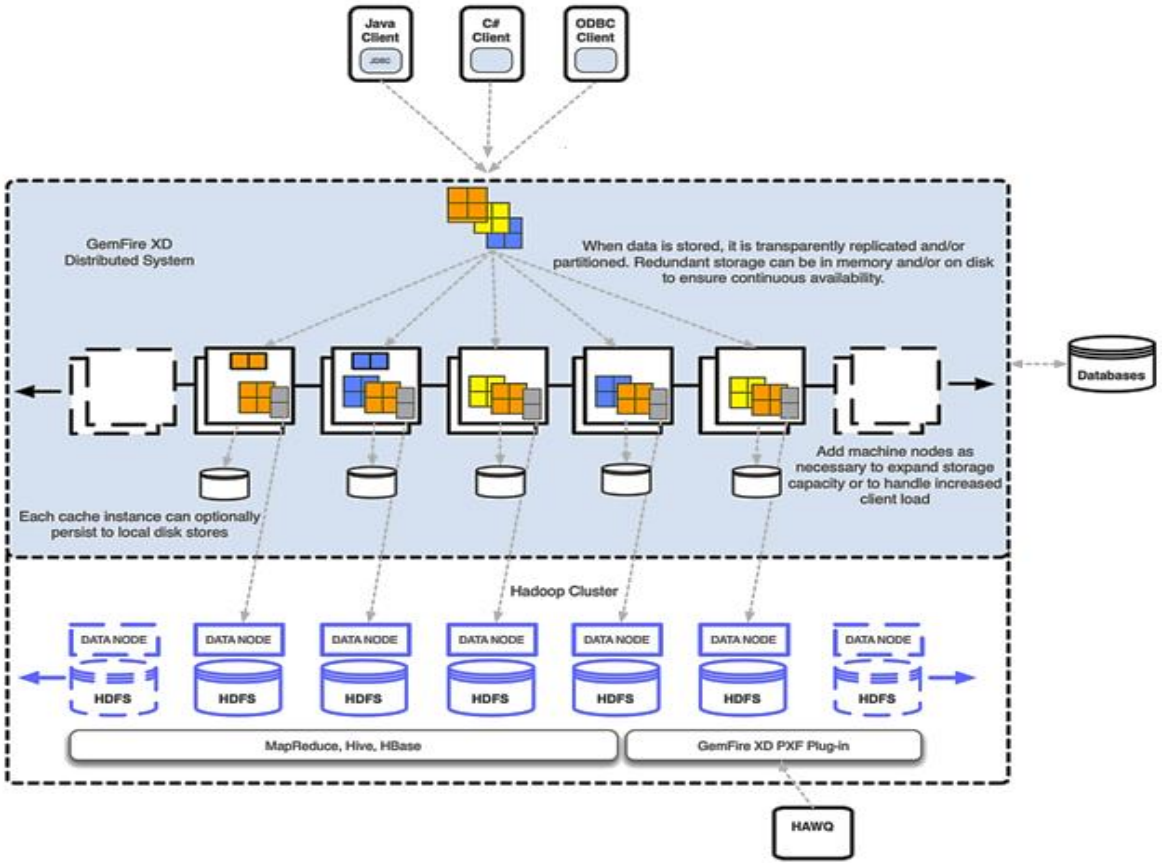


Figura 12 - Arquitetura Pivotal Gemfire IMDG Data Fabric, retirado de (Pivotal 2014).

O mesmo apresenta-se como uma plataforma 100% *open source* e a informação a seguir descrita refere-se às características do ecossistema do produto, retirado da documentação pública disponibilizada online [4].

[4] http://gemfire.docs.pivotal.io/

No Gemfire existe o conceito de regiões criadas dentro da *cache* como um agrupamento lógico para o conjunto de dados. Esse conjunto de dados é distribuído em memória, On-Heap e/ou Off-Heap, dentro das regiões existentes na *cache* criada, com o suporte da API JCache, que utiliza um modelo de dados de pares Chave/Valor. Ainda é possível definir a configuração de políticas de substituição (LRU) e de expiração de dados, bem como administrar e monitorizar o *cluster* de servidores de *cache* através de ferramentas da própria plataforma.

Uma característica importante fornecida pelo produto é a garantia de *cluster* resiliente e *failover*, pois tem um tempo de atividade contínuo que permite construir o cluster em alta disponibilidade, com recuperação de falhas. Isto é conseguido porque o sistema está preparado com modelos de deteção de falhas que permitem reagir rapidamente, garantindo que o *cluster* está sempre disponível e que o conjunto de dados não é perdido.

A interface SQL de baixa latência, chamada OQL (Object Query Language), permite ações de consulta que incluem *joins* e *aggregations* ao conjunto de dados na grid em memória. Ao nível de transações, esta garante as propriedades ACID no apoio a operações coordenadas, eficientes e seguras sobre os dados em memória em tempo real.

No que se refere à persistência dos dados na *cache* instanciada, é possível proceder à sua configuração opcional para garantir a durabilidade dos dados em caso de falha do nó. Este processo é executado com recurso à escrita, para o disco local, de um registo de criação, atualização e exclusão de dados no nó, que ao ser lido permite reconstruir o último estado consistente do mesmo em memória.

O suporte para o sistema de arquivos HDFS do Hadoop fornecido pelo Gemfire, permite armazenar e processar conjuntos de dados disponibilizados na *grid* em memória, manipulando-os posteriormente através das ferramentas Hadoop, como o Map Reduce, Hive e Pivotal HAWQ. O GemFire estende a API Hadoop Map Reduce para apoiar trabalhos de processamento paralelo em dados armazenados no HDFS, sem ter que para isso conectar-se ao sistema distribuído GemFire em memória. Ao contrário do GridGain, em que estes recursos são oferecidos num ecossistema integrado inteiramente na memória, o Gemfire por sua vez oferece os mesmos recursos, mas integrados no disco.

Por último, é importante referir que o Gemfire disponibiliza duas topologias de armazenamento de dados em cache, a Replicada e a Particionada com inclusão opcional de

persistência de dados (Jennings 2014). De notar que a caracterização das topologias e as referentes vantagens e desvantagens já foram mencionadas na subsecção 2.5.1.

No entanto, é ilustrada na Figura 13 referente à topologia Particionada, a execução de uma operação PUT por uma aplicação cliente, na escrita de um post "I like toast" na JVM1. Este armazena uma cópia primária na região do nó JVM2 e uma cópia redundante, que tem de ser pré-configurada, na região do nó JVM3.

A topologia Particionada Persistente, também ilustrada na figura, funciona de igual modo à Particionada, com a particularidade de cada nó do *cluster* armazenar o conjunto de dados para o disco local, de modo a ter maior persistência de dados. Isto poderá ser útil em situações em que o volume de dados seja superior à memória disponível ou para situações em que é desejável ter segurança adicional de persistência em caso de falha de um nó do *cluster*.

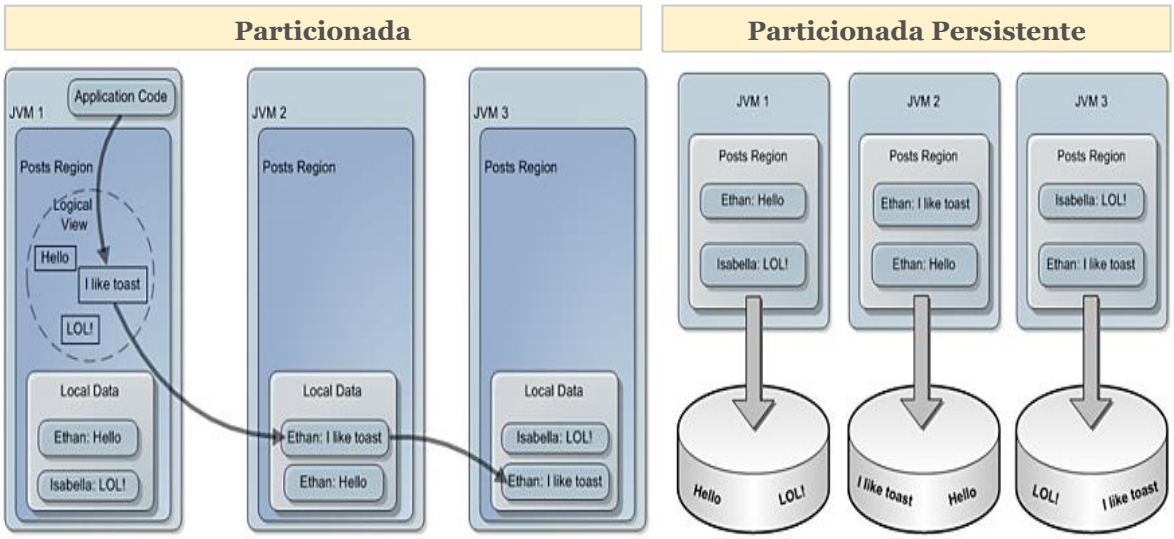


Figura 13 - Modo de Cache Particionada e Particionada Persistente, Pivotal GemFire, retirado de (Jennings 2014).

A Figura 14 ilustra um exemplo da topologia Replicada numa operação de PUT e GET a partir de uma aplicação cliente a diversos nós do *cluster*. Na operação de PUT, "Ethan" é escrito na região do nó JVM1 e replicado para os restantes nós JVM2 e JVM3. Na operação GET, quando é solicitado, "Ethan" é devolvido diretamente do nó JVM3.

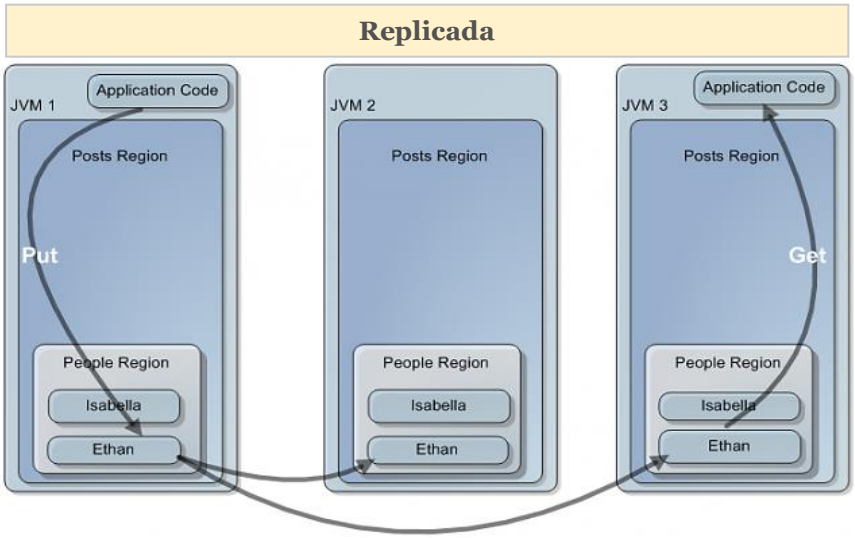


Figura 14 - Modo de Cache Replicada, Pivotal GemFire, retirado de (Jennings 2014).

2.7. Ferramenta de Testes Yardstick

A ferramenta Yardstick é um pacote de integração que permite escrever e realizar testes em sistemas distribuídos na memória para tecnologias IMDG. Esta oferece um conjunto de classes pré-desenvolvidas e propriedades de configuração que estão disponíveis após o carregamento da dependência no ficheiro *"pom.xml"* de cada pacote implementado, em servidores de *cache* e clientes, do projeto Java Maven. Este inclui um conjunto padrão de sondas que coletam métricas durante a execução dos testes, tais como:

- **ThroughputLatencyProbe:** mede a latência e taxa de transferência.
- **VmStatProbe:** recolhe informação fornecida pelo comando Linux vmstat, com várias métricas de utilização de CPU; Processos; Sistema; IO; Swap e Memória.

O Yardstick é suportado por um ficheiro de configuração, *"benchmark.properties"*, onde definimos sondas, JVM Tuning[5], endereçosIP da topologia e classes de teste. Nestas classes de teste podemos ainda definir um conjunto de parâmetros, tais como: ficheiro de configuração de

5 Configurações de JVM's para ajustar as máquinas ao nível de performance e estabilidade.

servidores de cache, tempo de execução, *warm-up*[6], número de *threads*[7], entre outros. A Figura 15 ilustra um exemplo do tipo de propriedades passíveis de configurar no ficheiro.

```
# JVM Tunning
JVM_OPTS=${JVM_OPTS}" \
  -Xloggc:./gc.log \
  -XX:+PrintGCDetails \
  -verbose:gc \
  -XX:+UseParNewGC \
  -XX:+UseConcMarkSweepGC \
  -XX:+UseTLAB \
  -XX:NewSize=128m \
  -XX:MaxNewSize=128m \
  -XX:MaxTenuringThreshold=0 \
  -XX:SurvivorRatio=1024 \
  -XX:+UseCMSInitiatingOccupancyOnly \
  -XX:CMSInitiatingOccupancyFraction=60 \
  -DgemfirePropertyFile=./config/geode.properties \
"

# Sondas
BENCHMARK_DEFAULT_PROBES=ThroughputLatencyProbe,VmStatProbe


# Servidores de Cache JVMs
SERVER_HOSTS=10.92.116.160, 10.92.116.161, 10.92.116.162

# Driver / Cliente de Execução de Teste
DRIVER_HOSTS=10.92.116.168

# Configuração do Teste
CONFIGS="\
-gfclicfg ${SCRIPT_DIR}/../config/geode-config.xml -nn ${nodesNum} -w 30 -d 60 -t 8 -cm -sp 15114 -dn
 GeodePUTGETXMLQueryBenchmark -sn GeodeNode -ds putgetBenchmark-WarmUp=30-Time=60-Threads=8,\
"
```

Figura 15 – Ficheiro de configuração "benchmark.properties".

Depois de configurada a topologia e as propriedades do teste a executar, basta utilizar os comandos *"./benchmark-servers-start.sh"* e *"./benchmark-drivers.sh"*, para executar os intervenientes individualmente ou em conjunto através de *"./benchmark-run-all.sh"* . Após execução dos testes, o Yardstick tem a capacidade de produzir relatórios com a informação recolhida pelas sondas, bem como ficheiros de registo, que posteriormente podem ser analisados e utilizados para gerar gráficos.

6 Tempo de aquecimento na execução de testes, com o intuito de obter resultados constantes.
7 Forma de um processo se dividir num conjunto de tarefas executadas concorrencialmente.

2.8. Trabalhos Relacionados

O trabalho apresentado em (Williams et al. 2014) descreve o estudo efetuado por uma organização multinacional de serviços e tecnologia, a General Electric, no desenvolvimento de um protótipo de infraestrutura utilizando tecnologias IMDG, para análise de grandes volumes de dados, na ordem dos TB, produzidos por sensores colocados numa frota de turbinas a gás, para ajudar na monitorização do desempenho dos equipamentos industriais, através de indicadores de performance (KPI). O produto utilizado para o desenvolvimento do protótipo foi o Pivotal Gemfire. Os dados são inseridos utilizando uma API baseada em Java e podem ser consultados utilizando uma linguagem SQL-like. Foram feitos testes exaustivos a outros produtos IMDG, como Hazelcast, Oracle Coherence, IBM's WebSphere eXtreme Scale, Terracota, mas concluiu-se que o Gemfire fornece um melhor desempenho e uma arquitetura mais simples do que as restantes. Ao nível de implementação, os dados eram provenientes de um arquivo que contém dados históricos que representam cerca de 2.000 turbinas a gás. Os dados foram inseridos através de um Cliente GemFire Java API na IMDG executada a um *cluster* dedicado. Os dados foram particionados por turbina e armazenados com replicação síncrona nas regiões da IMDG, para garantir segurança em caso de falhas de nós no *cluster*. Depois do armazenamento realizado, foram executadas consultas aos dados do servidor, pelo cliente, para recuperar o subconjunto de dados relevantes. As consultas foram escritas através da interface OQL (Object Query Language), linguagem SQL-like e utilizando índices especificados no arquivo de configuração, de modo a acelerar o desempenho da consulta. Diversos desafios foram encontrados no desenvolvimento deste protótipo. Em primeiro lugar, porque a maioria dos IMDGs utilizam o armazenamento de objetos tradicionais, e por isso, é fundamental identificar a estrutura certa para armazenar os dados de forma a maximizar o desempenho. Outro desafio importante foi estimar com precisão a quantidade de memória necessária para um determinado cenário. É importante saber antecipadamente a quantidade de dados que precisam de ser armazenados e a quantidade de memória que requerem esses dados, para que seja assegurada a quantidade de memória necessária atribuída ao IMDG.

De notar que quando trabalhamos com IMDGs na resolução de problemas, muitas vezes não é um processo simples e os resultados são altamente sensíveis consoante as

configurações do sistema, neste caso as JVM's. Este trabalho demonstrou que uma infraestrutura baseada em IMDG pode lidar com os requisitos de alto volume e alta velocidade no armazenamento e processamento de dados provenientes de sensores industriais, oferecendo taxas de desempenho elevado e ao mesmo tempo análise de dados em tempo real.

2.9. Síntese

O capítulo descreve o estudo de várias soluções existentes na área dos "Sistemas de Gestão e Armazenamento de Dados" numa perspetiva de acompanhar o crescimento do conceito *Big Data*. Algumas das soluções como, o NewSQL, o NoSQL e as IMDBMS visam responder a este crescimento, mas as tecnologias IMDG têm emergido neste contexto devido à sua plataforma extremamente completa e rica em recursos a partir da memória. Estas oferecem suporte para tecnologias Hadoop, como HDFS, Hive, Map Reduce, Storm, Impala, entre outros, e solucionam eficazmente a problemática do armazenamento e processamento distribuído de dados, oferecendo três requisitos importantes: Performance, Escalabilidade e Alta Disponibilidade.

Com base na similaridade dos produtos de mercado IMDG, selecionou-se o Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire para o desenvolvimento do protótipo de *software* à funcionalidade de *caching* de relatórios na plataforma NPM, bem como a ferramenta Yardstick, para a implementação do ambiente de testes, que nos capítulos seguintes são detalhados de forma mais aprofundada.

Capítulo 3

Arquitetura e Implementação

Este capítulo descreve o sistema atual Nokia Performance Manager, com ênfase no processo de caching de relatórios de performance, e a implementação do protótipo funcional proposto neste trabalho usando duas soluções baseadas em IMDG.

Primeiramente, na secção 3.1, é apresentada a topologia atual da ferramenta NPM, no que se refere a interfaces de utilizador, modelo 3GPP e descrição de granularidades por objeto e tempo. Nesta secção ainda é detalhada a implementação atual da arquitetura utilizada no NPM com recurso a uma base de dados ORACLE.

Na secção 3.2, é descrita a implementação da solução em tecnologias *IMDG*. Inicialmente é clarificado o que pretendemos desenvolver e posteriormente é detalhado ao nível de desenvolvimento o diagrama de classes, a estrutura de dados, o mecanismo de consulta de dados para fazer a ligação com a interface de utilizador, a arquitetura do sistema e por último a configuração da *cache* dos servidores para Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire.

3.1. Estado Atual do Sistema

3.1.1. Topologia Nokia Performance Manager

Atualmente, a funcionalidade de *caching* ORACLE NPM está direcionada para o consumo de relatórios de desempenho que são solicitados frequentemente. Estes relatórios podem ser pré-calculados em determinados intervalos e são imediatamente disponibilizados quando requisitamos o relatório. Podemos ter intervalos por granularidade de tempo (hora, dia, mês, ano) e por granularidade de objeto (PLMN, MRBTS, LNBTS e LNCELL), como podemos visualizar

nas Interfaces de Utilizador, na Figura 16. Estes objetos são designados por identificadores no modelo hierárquico *3rd Generation Partnership Project* (3GPP).

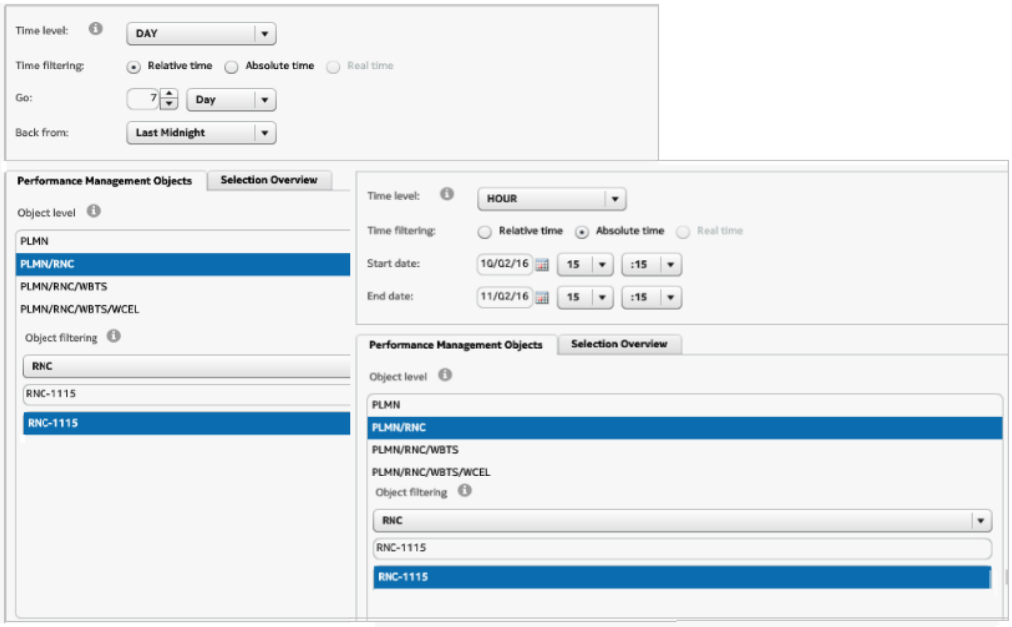


Figura 16 – Interfaces de Utilizador (UI) para *Caching* de Relatórios ORACLE.

O modelo 3GPP representa uma organização tecnológica do ramo das telecomunicações, que visa padronizar, criar, enviar e reproduzir arquivos em telefones e aparelhos *wireless* GSM em redes 2G, 3G e LTE (Tel 2016). A Figura 17 representa a hierarquia de classes de objetos para a tecnologia LTE com o mapeamento para a nomenclatura utilizada na Nokia.

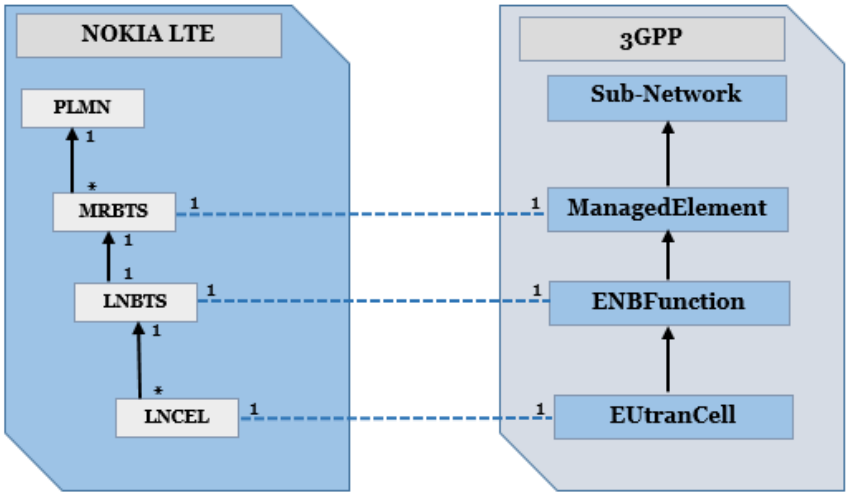


Figura 17 – Mapeamento para a Nomenclatura Nokia LTE.

O objeto PLMN (Public Land Mobile Network) é o sistema de comunicações que representa toda a rede do operador, identificada pelo código da rede móvel e do país. O objeto MRBTS (Multi Radio Base Transceiver Station) é a estação física, representada pela tecnologia utilizada, que no caso das BTS da Nokia é o 4G ou LTE. O objeto LNBTS (Long-Term Evolution Node Base Transceiver Station) representa um conjunto de células em comunicações móveis de 4G ou LTE. O objeto LNCELL (Long-Term Evolution Node Cell) representa uma célula associada a uma área geográfica coberta por uma determinada antena em comunicações móveis de 4G ou LTE. Posto isto, a seleção dos fatores objeto e tempo a partir da interface de utilizador vai permitir realizar um conjunto de consultas que serão processadas de modo a gerar o relatório pretendido.

3.1.2. Implementação ORACLE

Ao nível funcional e de implementação, o ORACLE NPM é essencialmente utilizado como um armazém centralizado de dados. A arquitetura implementada no NPM é composta por um conjunto de processos e camadas que compõem todo o sistema utilizando o ORACLE como base de dados central. Temos portanto, a camada de "Fonte de Dados", "Mediador", "Processamento" e "Apresentação", com diversos módulos, como verificado na Figura 18.

Na camada *"Fonte de Dados"*, temos um sistema de ETL (Extração, Transformação, Carga), responsável por colecionar os dados, normalmente em ficheiros com formatos díspares (XML, ASCII, CSV).

A camada *"Mediador"* é responsável por iniciar a fase de conversão, onde esses ficheiros são convertidos para um formato XML, com um esquema proprietário da Nokia, denominado por OMeS (Open Measurement Standard).

Seguidamente, na camada de *"Processamento"*, existe um módulo que carrega esses ficheiros XML no ORACLE, a que chamamos de PM (Performance Manager) pipe. Para além de carregar os dados em tabelas do PM, ou em nomenclatura DW, em tabelas de factos, insere também entradas em tabelas de topologia, ou em termos de DW, em tabelas de dimensões, de forma a registar todos os objetos que entram no sistema. Aqui, um objeto é equivalente às instâncias de PLMN, MRBTS, LNBTS, LNCELL, entre outras.

Após os dados serem carregados para a base de dados ORACLE, existem processos agendados (hora, dia, mês, ano) que vão calcular agregações sobre os dados originais, os chamados dados *raw*. A ideia das agregações é sobretudo otimizar a performance de *reporting*, ou seja, imaginemos que alguém solicita um relatório diário sobre todas as MRBTS. Ao invés de calcular um valor por dia por MRBTS a partir dos dados *raw* que foram carregados e que ascendem a milhares de entradas, podemos ir buscar os dados já agregados ao dia por MRBTS a uma outra tabela de agregados. A utilização de módulos de agregações permite que não seja necessário guardar os dados *raw* durante tanto tempo, uma vez que temos dados agregados ao dia, semana, mês já comprimidos, podendo assim poupar em custos de armazenamento.

Por último, a camada de "*Apresentação*" ou *reporting* que é onde se constroem e executam os relatórios via interface de utilizador, utiliza uma série de serviços para suportar essas atividades, como o mecanismo de consulta. Este é responsável por interpretar as seleções do utilizador relativamente a parâmetros de relatório (seleção tempo e objeto) e gerar uma consulta de SQL que possa ser executada diretamente em cima do ORACLE, via JDBC, fazendo uso de tabelas de PM, raw ou agregadas, dependendo de onde estiverem os dados necessários para gerar o relatório.

Após executar cada relatório, com todas as consultas necessárias para suportar o mesmo, o resultado, antes de ser enviado para a interface de utilizador, é armazenado na *cache*. Essa *cache* atualmente é implementada com recurso a uma tabela do ORACLE, que simula algo do tipo pares de Chave/Valor, onde se guarda o resultado do relatório. Da próxima vez que o relatório for solicitado, onde os parâmetros não mudam, é obtido diretamente da cache. Este processo é bastante mais rápido porque normalmente um relatório de PM utiliza diferentes grupos de contadores ou medidas que são guardados em diferentes tabelas de PM. Isto significa que quando queremos misturar tudo no mesmo relatório temos de fazer *joins* entre as tabelas de PM. A degradação em termos de tempos de execução é quase diretamente proporcional ao número de *joins* e tipicamente são utilizados entre 4-12 *joins* em cada relatório gerado. Em cima disto, ainda temos *joins* com as tabelas de dimensão, onde vamos buscar os nomes de objetos e informação adicional.

Posto isto, é bastante mais rápido se guardarmos o resultado do relatório após correr as diversas consultas numa tabela e obter o relatório daí, da próxima vez. É apenas necessário fazer

uma seleção a essa tabela pela chave do relatório, chave essa que é constituída por um *hash* com todos os parâmetros do relatório. De notar, que o tempo de vida dos objetos ou relatórios na *cache* é normalmente de 48 horas.

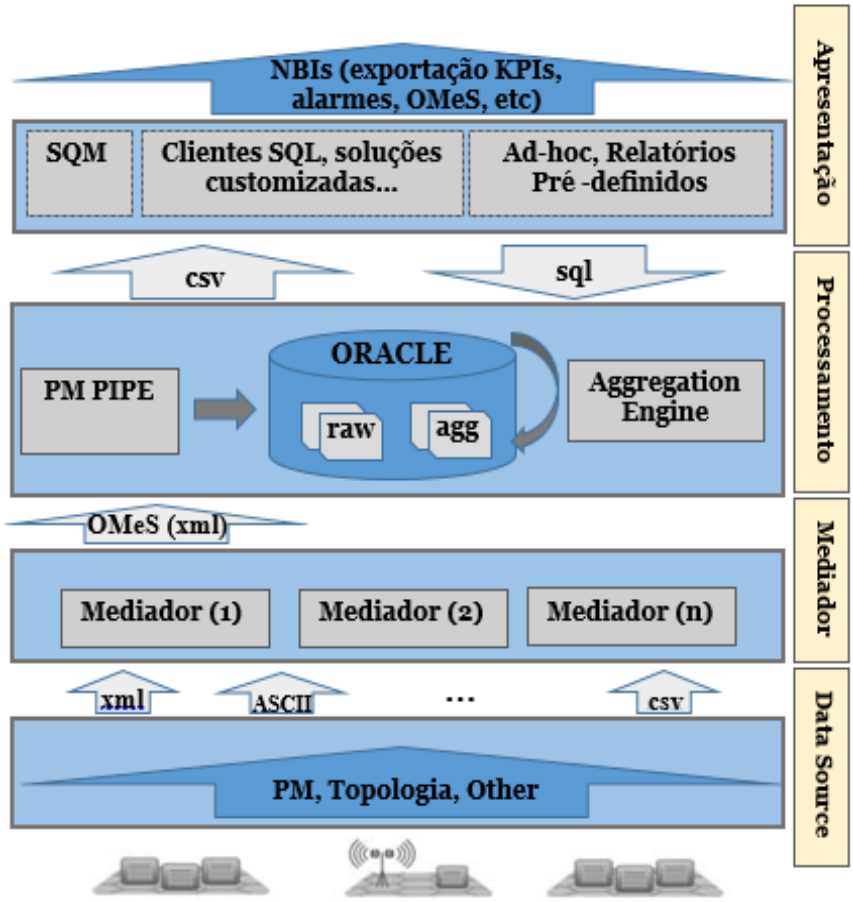


Figura 18 – Arquitetura Corrente Nokia.

O processo de *caching* de relatórios no NPM é um dos fatores mais críticos desta arquitetura, devido à performance e limitação da cache ORACLE atual, relacionada com o processamento de toda a informação registada pelos aparelhos de rede, que processam milhões de registos, com tendência para crescer. Por padrão, todas as execuções de relatórios são armazenados na cache da base de dados ORACLE. Isto significa que todas as execuções subsequentes do mesmo relatório irão fornecer uma resposta mais rápida e ao mesmo tempo vai evitar o processamento de dados desnecessários e a utilização de recursos.

O panorama do sistema atual NPM é basicamente, se o relatório num dado intervalo de tempo para um dado tipo de objeto já tiver sido gerado, então é devolvido diretamente da *cache*

da base de dados ORACLE, ao utilizador. Se o relatório num dado intervalo de tempo para um dado tipo de objeto não tiver sido gerado, então esse relatório terá que ser gerado com recurso à base de dados ORACLE em disco e posteriormente armazenado na *cache*. O processo de armazenamento e geração de relatórios no NPM está bem explícito na Figura 19.

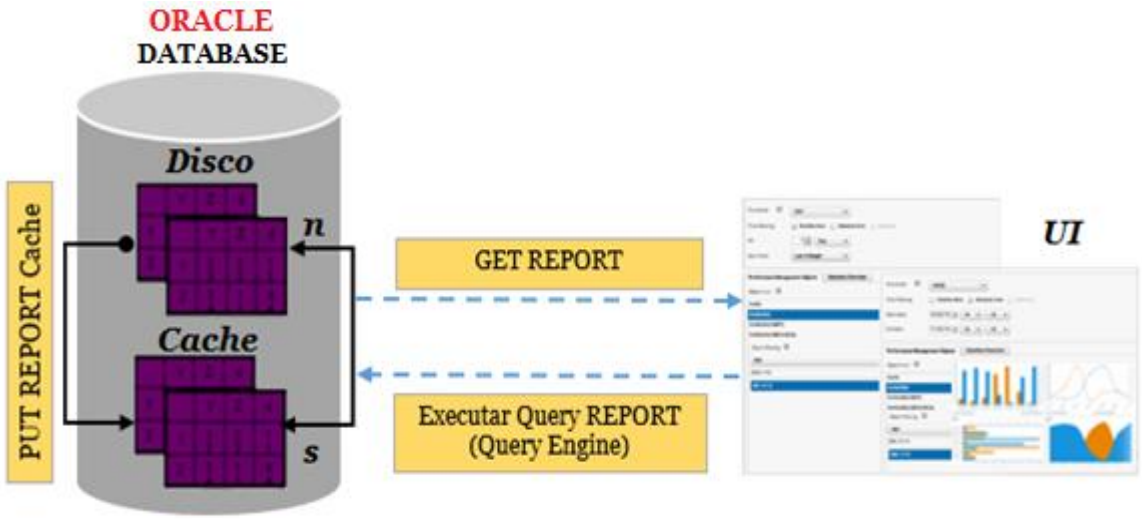


Figura 19 – Processo de Caching Report Oracle NPM.

3.2. Solução com Grid de Dados em Memória

3.2.1. Requisitos

Tendo por base o processo de *caching* de relatórios de rede na plataforma NPM, o que se pretende fazer é o desenvolvimento e configuração de uma arquitetura (Cliente-Servidor) distribuída totalmente em memória, com recurso aos produtos IMDG, o Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire, para o armazenamento e processamento intensivo de relatórios de performance de rede.

Ao invés de utilizar o sistema de cache ORACLE, atualmente implementado, pretendemos desenvolver uma nova solução fazendo uso de tecnologias IMDG. O processo de armazenamento em *cache* vai ser muito similar ao já existente, mas com a particularidade de utilizar um conjunto de servidores de *cache* que armazenam os dados em memória. A Figura 20 representa o fluxo do protótipo de *software* implementado, que é similar ao já descrito na secção 3.1.2.

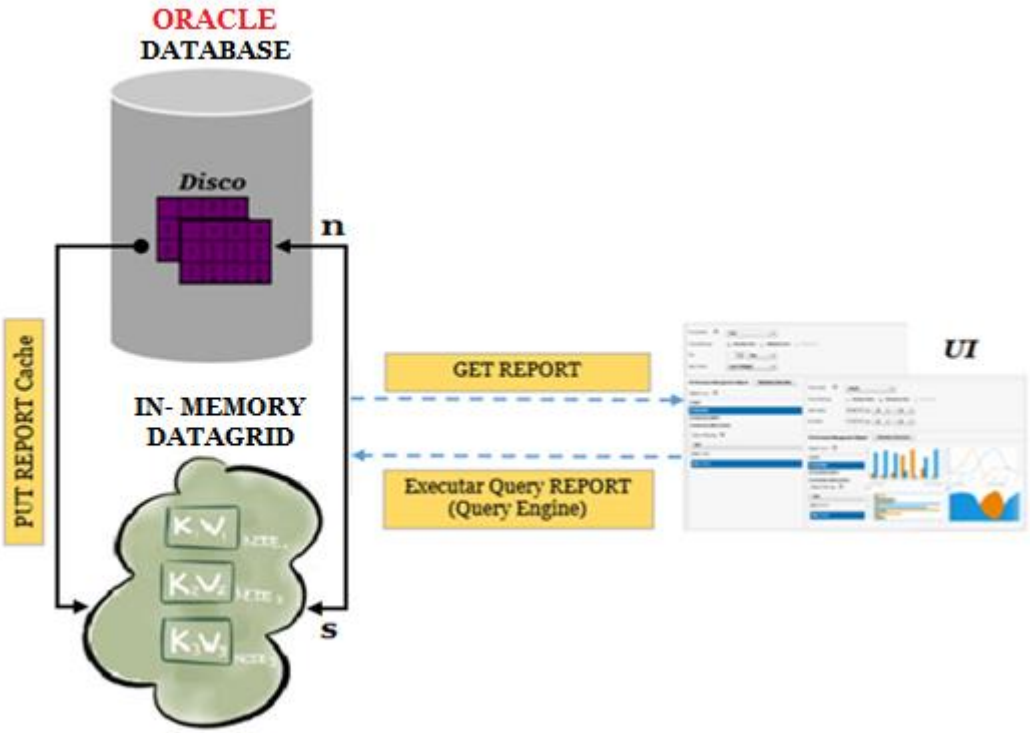


Figura 20 - Processo de Caching Report IMDG NPM.

Como a ideia é implementar um protótipo funcional de geração de relatórios, simulando um conjunto de seleções por granularidade de objeto e tempo, consoante a escolha do utilizador, foi implementada uma interface JAVA de raiz, com base na utilizada na ferramenta NPM, como podemos observar na subsecção 3.1.1. Esta vai permitir requisitar relatórios, devolvê-los da *cache*, caso existam, ou gerar um novo relatório e armazená-lo na *cache* posteriormente.

Para além disto, é essencial definir uma topologia adequada para a arquitetura, a respetiva modelação de dados sob uma estrutura de pares Chave/Valor, bem como proceder à configuração dos servidores de cache e respetiva interligação com a interface de utilizador.

Seguidamente, preparamos um ambiente de testes de carga aos servidores de cache, numa arquitetura configurada na Nokia e analisamos operações de PUT e GET de relatórios simultaneamente, por forma a perceber o comportamento da cache do novo sistema.

No final estaremos em condições de concluir, com base nos resultados gerados, o melhor produto IMDG, para os casos de testes implementados, no processo de *caching* de relatórios de rede, numa possível remodelação do processo na ferramenta NPM.

3.2.2. Diagrama de Classes

O protótipo funcional foi desenvolvido com recurso à linguagem de programação JAVA Maven, com declaração de dependências em *"pom.xml"*, e testado em ambiente Linux, sobre a utilização de um ambiente virtualizador para a criação de máquinas virtuais. Estas vão ser utilizadas para instalar e configurar os pacotes implementados, sendo que algumas máquinas vão funcionar como *servidores de cache* e apenas uma máquina como *cliente*, simulando a comunicação cliente-servidor.

De notar, que os pacotes das tecnologias Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire foram desenvolvidos de raiz, de forma independente e com a mesma estrutura com o intuito de podermos no final avaliá-las sobre o mesmo caso de estudo.

O modelo de classes construído para ambas as tecnologias pode ser analisado mais pormenorizadamente com as ilustrações apresentadas nas Figuras 21 e 22 respetivamente, contendo as seguintes classes:

Cliente

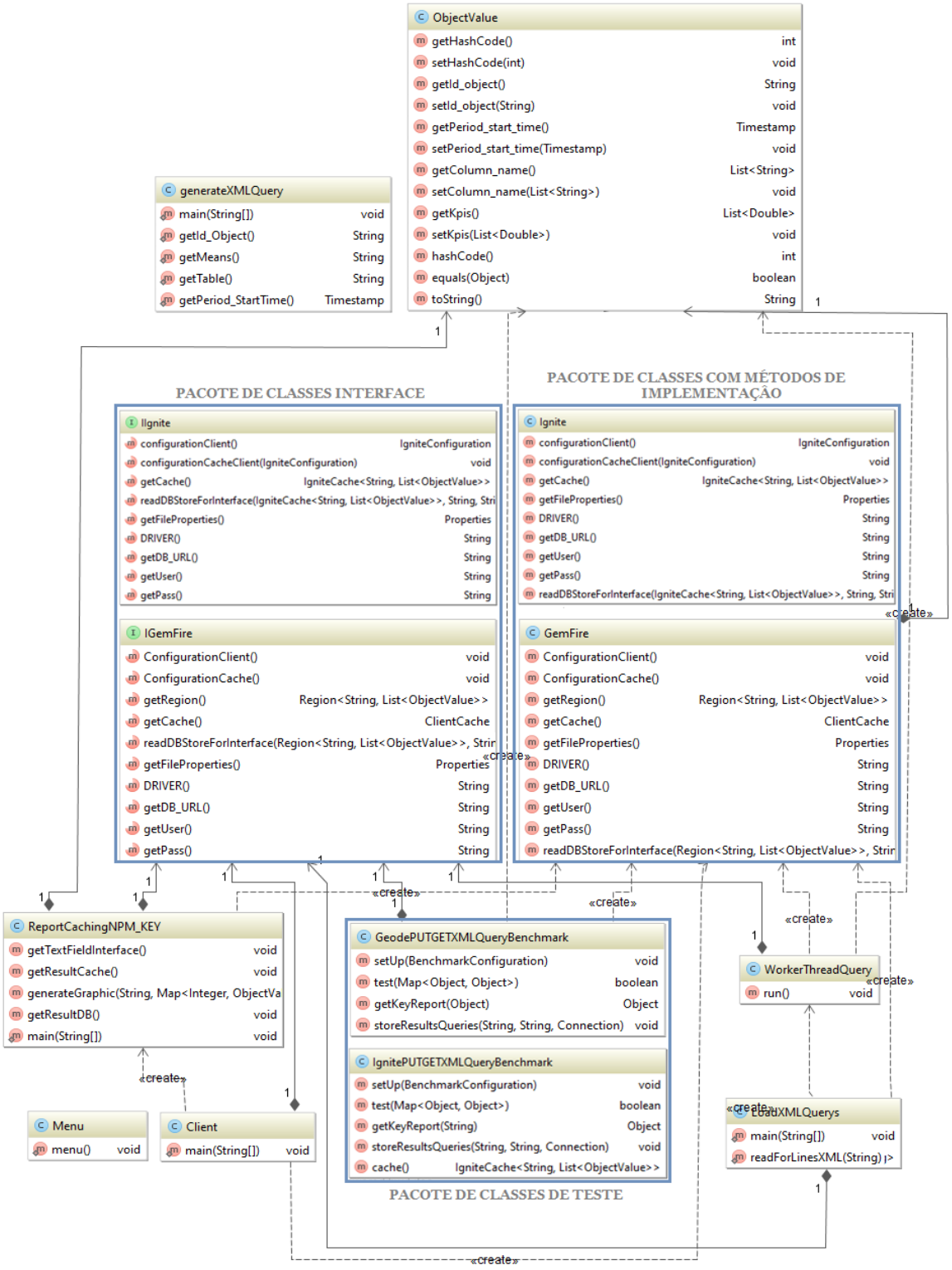


Figura 21 – Diagrama de Classes *Cliente* Ignite e Gemfire.

- Client.java: classe de inicialização e comunicação do cliente aos respetivos servidores de *cache* e interface de utilizador para a funcionalidade de *caching* de relatórios.
- IIgnite.java e IGemFire.java: interfaces responsáveis pelos métodos e propriedades essenciais às classes "Ignite.java" e "Gemfire.java".
- Ignite.java e GemFire.java: classes que implementam as interfaces acima, com métodos de configuração de cliente, cache, ligações jdbc e leitura de propriedades (endereçoIP, portas, base de dados) específicas do ficheiro "config.properties".
- ReportCachingNPM_KEY.java: classe interface de utilizador que simula o *caching* de relatórios solicitados pelo utilizador. Esta contém métodos de desenvolvimento gráfico e validação da utilização de *cache* ou base de dados consoante a existência de relatórios.
- LoadXMLQuerys.java: classe responsável por carregar um conjunto extenso de consultas provenientes de um ficheiro XML, com um dado formato.
- WorkerThreadQuery.java: classe que testa a utilização de *threads* em operações PUT à *cache* a partir da base de dados, recorrendo às consultas carregadas na classe anterior.
- GenerateXMLQuery.java: classe independente para a geração de ficheiros XML com consultas aleatórias sobre uma dada estrutura.
- ObjectValue.java: classe responsável pela modelação dos objetos de dados.
- GeodePUTGETXMLQueryBenchmark.java/ IgnitePUTGETXMLQueryBenchmark.java: classes implementadas para os testes de carga ao protótipo de *software*.

Servidores de *Cache* e Localizador

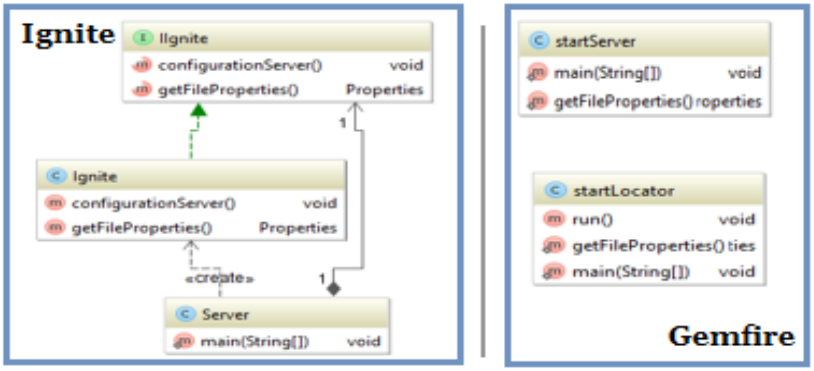


Figura 22 – Diagrama de Classes *Servidores de Cache e Localizadores* Ignite e Gemfire.

- Server.java e StartServer.java: classe de inicialização do servidor e respetiva comunicação a outros nós servidor, com recurso a métodos específicos de configuração e leitura de propriedades.
- StartLocator.java: classe de inicialização do localizador ou mediador excecionalmente na tecnologia Gemfire.

É importante referir que as propriedades de configuração de endereços IP, portas e ligações via jdbc, são efetuadas no ficheiro "config.properties" de cada pacote, permitindo a comunicação entre os agentes envolvidos, os servidores de cache, o cliente e a base de dados.

Para testar o funcionamento dos pacotes basta executar via linha de comandos (**ex:** java –jar ./GridGainServer/target/GridGainServer-1.0-SNAPSHOT-jar-with-dependencies.jar) os pacotes JAR gerados de cada servidor de *cache* e cliente para verificar a comunicação automática entre os nós da arquitetura.

3.2.3. Estrutura Chave/Valor

As tecnologias IMDG permitem fazer a modelação de dados com recurso a estruturas de pares Chave/Valor, onde a chave é o identificador exclusivo para o conjunto de valores associado.

O objetivo do processo de *caching* de relatórios é o armazenamento de relatórios de dados gerados na rede, e como tal, esta tem um conjunto de elementos ou variáveis que necessitam de ser armazenados em cache, para que possam posteriormente ser mostrados ao utilizador. Estes elementos são, o *id_object,* identificador do nível do objeto (PLMN/MRBTS/LNBTS/LNCEL), o *period_start_time*, período de tempo em que o dado foi registado na rede, o *column_name* e *kpis,* valores de performance registados na rede para cada coluna definida.

A estrutura de dados foi modelada com recurso a uma chave, designada por *report key*, uma *String* que concatena os vários elementos da pesquisa na interface de utilizador. Por exemplo, se o utilizador quer gerar um relatório para o *id_object*, "PLMN-PLMN/MRBTS-1", para um *period_start_time* entre "01-01-2016 00:00:00" e "01-01-2016 01:00:00", agregado por "hour" para uma determinada medida "X", então o *report key* a ser usado como estrutura é **<PLMN-PLMN/MRBTS-1#01-01-2016 00:00:00#01-01-2016 01:00:00#hour#X>**.

Associado a esta chave existe o valor que é fundamentalmente uma lista de objetos definida por *ObjectValue*, que contém a informação relativa ao relatório. O *ObjectValue* armazena um *hashCode (INT)*, o identificador do registo de rede, o *id_object* (STRING), o *period_start_time* (TIMESTAMP), a *column_name* (LIST <STRING>) e os *kpis* (LIST <DOUBLE>).

A Tabela 3 exemplifica a estrutura que necessitamos para armazenar a informação na *cache*, sendo que, se o utilizador solicitar uma dada chave do relatório e esta existir em *cache*, a lista de objetos para essa chave será automaticamente retornada na interface de utilizador, caso contrário o relatório terá que ser construído e armazenado na cache sob a estrutura ilustrada, recorrendo à base de dados existente.

Chave (String)	**Valor (List<ObjectValue: {hashCode, id_object, period_start_time, List<String>, List<Double>}>)**
"PLMN-PLMN/MRBTS-1#01-01-2016 00:00:00#01-01-2016 01:00:00#hour#X"	{"1, PLMN-PLMN/MRBTS-1, 01-01-2016 00:30:00, List<A, B, C>, List<20.0, 25.0, 49.0>"} {"2, PLMN-PLMN/MRBTS-1, 01-01-2016 00:45:00, List< A, B, C >, List<50.0, 55.0, 69.0>"}
"PLMN-PLMN/MRBTS-2#01-01-2016 00:00:00#02-01-2016 00:00:00#day#Y"	{"1, PLMN-PLMN/MRBTS-2, 01-01-2016 00:33:00, List<A, B, C>, List<10.0, 15.0, 39.0>"} {"2, PLMN-PLMN/MRBTS-2, 01-01-2016 10:45:00, List< A, B, C >, List<60.0, 57.0, 86.0>"} {"3, PLMN-PLMN/MRBTS-2, 01-01-2016 23:55:00, List< A, B, C >, List<50.0, 55.0, 69.0>"}

Tabela 3 – Estrutura Chave/Valor Caching Report.

De notar, que antes de serem armazenados os registos da rede no objeto *ObjectValue*, este é serializado e desserializado recorrendo à interface *java.io.Serializable*. Esta permite armazenar o objeto na cache, capturando o estado de um objeto, tornando-o persistente e eficiente nas consultas ao objeto específico.

Em modo de exemplo, depois de solicitar os resultados existentes na cache, obtém-se o resultado ilustrado na Figura 23.

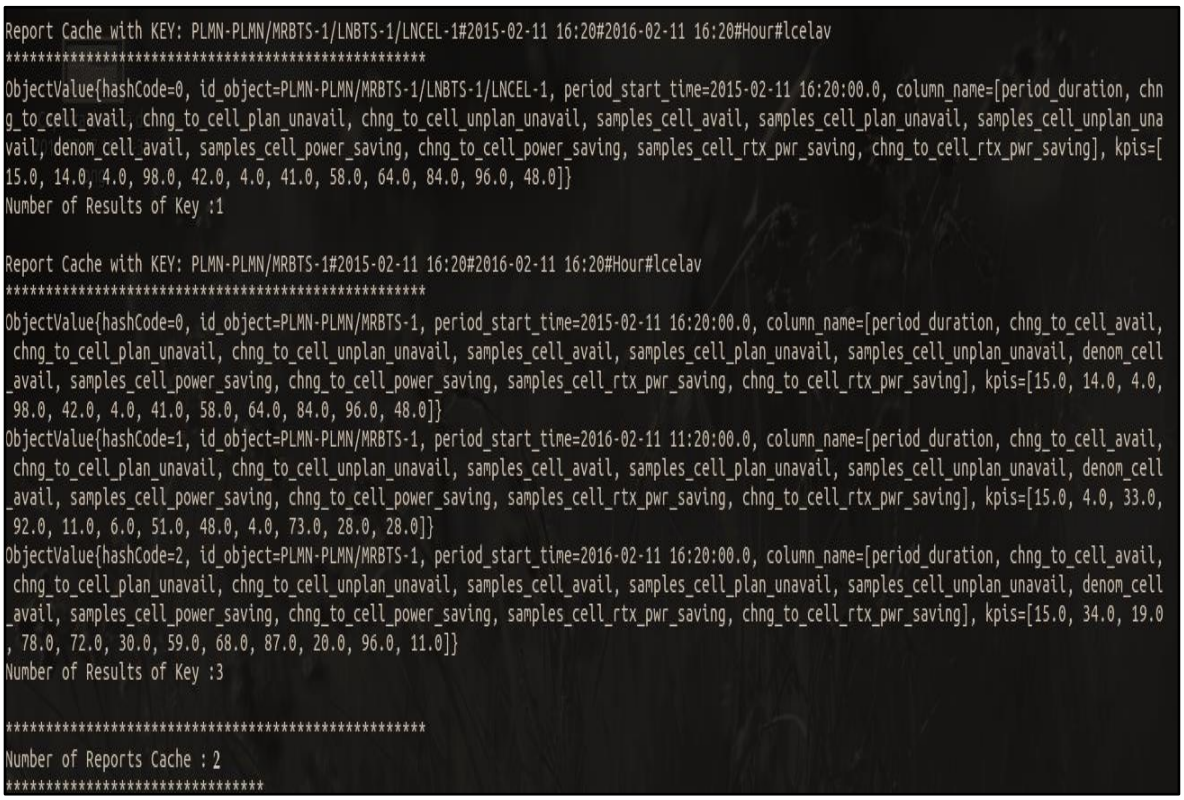


Figura 23 – Exemplo de GET aos relatórios nos Servidores de Cache IMDG.

3.2.4. Mecanismo de Consulta

O mecanismo de consulta desenvolvido é responsável por interpretar as seleções do utilizador, via interface de utilizador, relativamente a parâmetros de relatório, tais como: o nível de objeto; o período de tempo; o nível de agregação.

Este vai gerar uma chave, denominada por *"reportKey"*, e valida a sua existência na *cache*, ou caso não exista, gera uma consulta SQL e faz a construção do relatório a partir da base de dados, via jdbc, armazenando-o e retornando-o da *cache* posteriormente.

A Figura 24 ilustra a interface de utilizador desenvolvida com um exemplo de solicitação de um relatório com os parâmetros ao nível de objeto, "PLMN-PLMN/MRBTS-1", período de tempo entre "11-02-2016 16:00:00" e "11-02-2016 17:00:00" e agregado por "Hour".

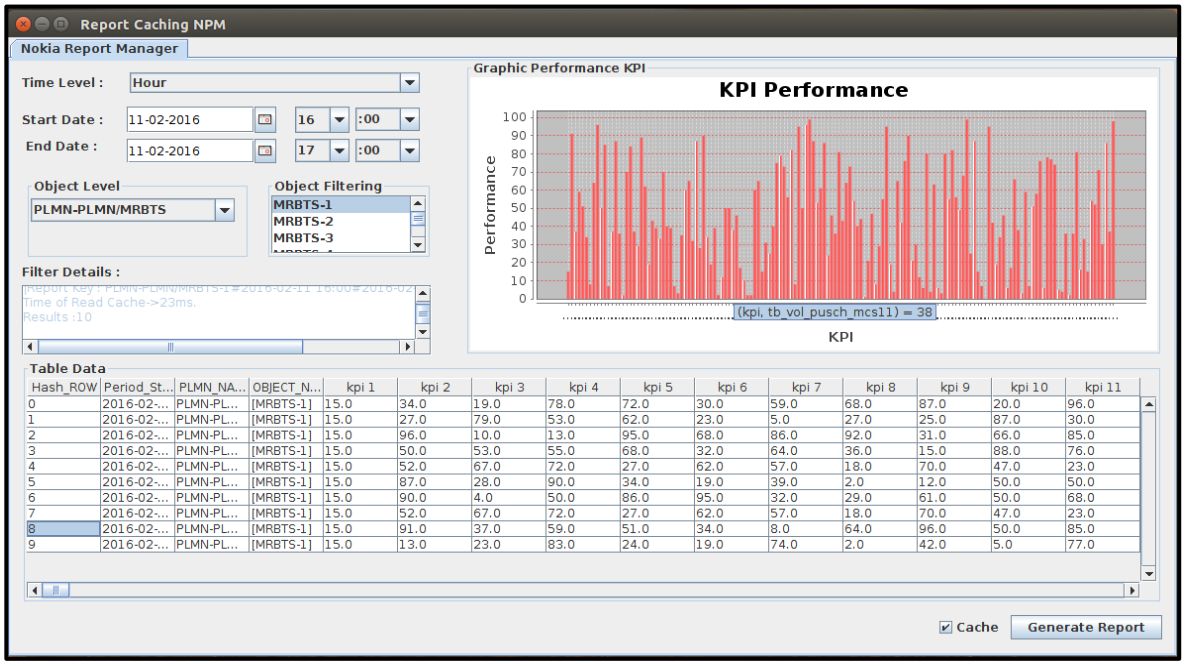


Figura 24 – Interface UI para Caching Report IMDG.

Por configuração, o relatório será devolvido da *cache*, através da construção da chave utilizando os parâmetros selecionados pelo utilizador, que resultará algo desta forma:

- **<PLMN-PLMN/MRBTS-1#11-02-2016 16:00:00#11-02-2016 17:00:00#hour#X”>**

Seguidamente, o método de validação verifica se a *cache* contém a chave acima descrita. Caso esta chave não exista, é solicitada a utilização da base de dados para gerar o relatório requisitado. Então, procedemos à construção de uma consulta através dos mesmos parâmetros e armazenamos o relatório na *cache*, devolvendo-o ao utilizador. A consulta com os parâmetros solicitados geram algo desta forma:

- **[SELECT * FROM noklte_p_lcelav_mrbts_hour a where (a.mrbts_id='PLMN-PLMN/MRBTS-1') AND (a.period_start_time >='2016-02-11 16:00' AND a.period_start_time <='2016-02-11 17:00')]**

Depois de devolvido o relatório solicitado, é apresentada na interface de utilizador uma lista dos objetos com os valores de performance da rede para a chave, denominado por kpis, que sendo individualmente selecionados formam um gráfico com esses mesmos valores, para que possamos visualizar e analisar de forma mais clara o relatório.

3.2.5. Arquitetura do Sistema

A arquitetura IMDG para Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire implementada no protótipo sugere a interação entre *clientes* e *servidores de cache*, comunicando-se através de uma rede de máquinas distribuídas. O cliente interage normalmente através de uma interface de utilizador e tem a função de iniciar pedidos, esperar e receber respostas e conectar-se de uma só vez a um conjunto de servidores. O servidor espera por pedidos, responde com informação solicitada e comunica com outros servidores para atender a solicitações específicas do cliente.

A Figura 25 ilustra o modelo de arquitetura implementado sob diversas camadas, tais como: servidores de cache; mediador; cliente; base de dados.

Temos portando, um conjunto de máquinas ou servidores virtuais, com os mesmos recursos de memória, disco e CPU, geridos por um ambiente virtualizador, que vão ser configurados como *servidores de cache* utilizando uma topologia adequada, para atender a operações de PUT e GET de relatórios sempre que os clientes requisitarem o serviço.

A comunicação entre Cliente e Servidor é efetuada através de um mediador de pedidos e respostas, cujo objetivo é a descoberta e comunicação com todos os nós da topologia, através dos endereçoIP e respetivas portas. O cliente é configurado para comunicar com os nós servidor por intermédio do mediador. Cada cliente tem um pacote de funcionalidades com operações de PUT e GET aos servidores de cache, via interface de utilizador e/ou consola de linha de comandos, bem como a respetiva ligação à camada de base de dados.

Esta ligação oferece a comunicação à base de dados PostGresSQL e a um *cluster* de dados distribuídos, armazenados no sistema de arquivos HDFS do Hadoop, com mecanismos de processamento Map Reduce e MPP *(Processamento Paralelo Massivo),* para Hive e Impala, em consultas SQL avançadas.

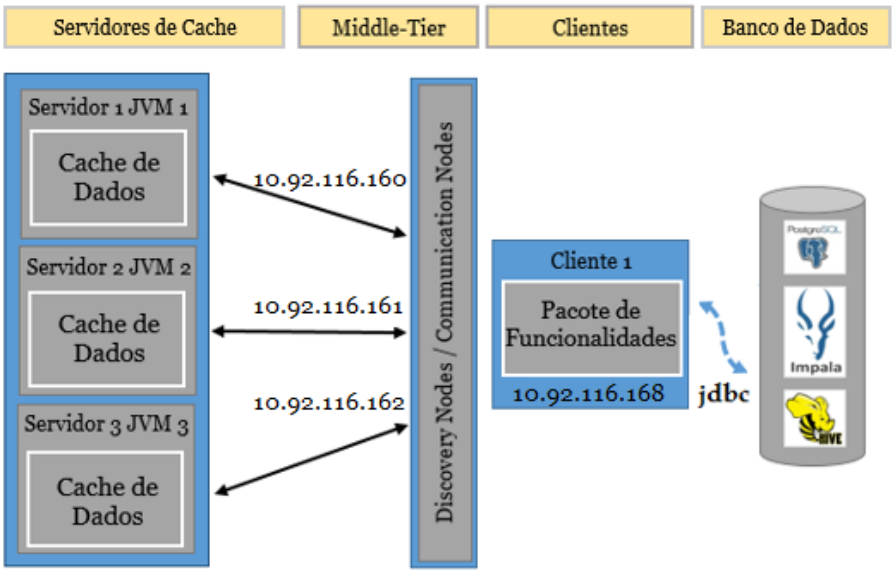


Figura 25 – Arquitetura Cliente/Servidor IMDG.

3.2.6. Configuração Apache Ignite/GridGain

Após ilustrar a arquitetura IMDG que pretendemos implementar no protótipo de *software*, apresenta-se a configuração dos *servidores de cache*, *cliente* e *topologia de cache* adotada para Apache Ignite/GridGain. Para ter acesso aos métodos e propriedades oferecidas pela tecnologia, é essencial carregar a dependência principal no ficheiro *"pom.xml"* de cada pacote do projeto Java Maven.

No que diz respeito ao desenvolvimento do *cliente e servidores de cache*, a Listagem 1 no Anexo A lista o código de execução da topologia do sistema. Os pacotes independentes são implementados de forma muito similar, com a particularidade de definir ao nível de cliente uma propriedade (*Ignition.setClientMode (true)*), que faz a distinção entre os agentes ativos envolvidos. Na função de configuração chamamos um outro método (*getFileProperties ()*), cujo objetivo é fazer a leitura das propriedades definidas no ficheiro *"config.properties"*, com endereçosIP, portas, entre outros. Estas propriedades são utilizadas para fazer a comunicação de toda a topologia distribuída. O Apache Ignite fornece métodos específicos para comunicar e descobrir nós na rede, através do *"TCPDiscoveryVmIPFinder"*, que ao associar uma lista pré-configurada de endereçosIP e portas, permite a descoberta automática de conexões entre os nós

servidores de *cache* e clientes. Após execução dos programas individualmente, a comunicação é estabelecida e teremos diversos nós ligados uns aos outros.

A configuração da *topologia de cache* em memória é fundamental para armazenar dados nos nós dos servidores de *cache*, sob um conjunto de propriedades e características fornecidas pela própria tecnologia.

A Listagem 2 no Anexo A exemplifica a criação da cache programaticamente, denominada por *"reportCacheNPM"*, com a estrutura de dados documentada na secção 3.2.3.

Perante os requisitos de configuração da mesma, definimos um modo de *cache "PARTITIONED"*, em que o conjunto de dados totais é igualmente dividido em partições nos nós do *cluster dos* servidores disponíveis. Este modo já oferece por si a disponibilidade de dados, mas mesmo assim utilizou-se uma propriedade de *"backup"* para redundância dos dados, permitindo armazenar uma cópia de segurança de cada chave na *grid* de dados, evitando a perda dos mesmos em caso de falha nos nós. O modo de memória utilizado na *cache* é o *"ONHEAP_TIERED"*, onde todas as entradas são armazenadas em *on-Heap*, sendo controladas por uma política de substituição que controle o número máximo de entradas que podem ser armazenados na *cache* em memória. A política utilizada é o LRU *(Least Recently Used),* que garante que as entradas menos utilizadas recentemente sejam substituídas em primeiro lugar. Para finalizar, foi definida uma política de expiração de entradas na cache, garantindo a sobrevivência das mesmas num dado período de tempo.

A tecnologia é extremamente rica em propriedades de configuração de *cache*, o que permite selecionar e configurar as que mais se adequam ao tipo de *cache* e de gestão que pretendemos, como podemos visualizar na Tabela 4.

Propriedade	Descrição
Modo de Atomicidade	
ATOMIC	Semântica atómica, consistência de dados, melhor desempenho.
TRANSACTIONAL	Semântica transacional, totalmente ACID, desempenho lento.
Modo de Cache	
PARTITIONED	*Cache* distribuído, conjunto de dados divididos igualmente entre os nós disponíveis, altamente escalável.

REPLICATED	*Cache* replicada, conjunto de dados são replicados para todos os nós do *cluster*, oferecendo máxima disponibilidade.
Modo de Escrita de Entradas na *Cache*	
FULL_SYNC	O cliente vai esperar pela escrita e atualização completa em todos os nós remotos (primários e de backup).
FULL_ASYNC	O cliente não espera pela escrita e atualização completa em todos os nós remotos (primários e de backup).
PRIMARY_SYNC	O cliente vai esperar pela escrita completa no nó primário, mas não espera que o nó de *backup* seja atualizado.
Modo de Memória	
ONHEAP_TIERED	Entradas na *cache* são armazenadas em *on-heap*. Estas podem ser movidas para *off-heap* quando atingir um máximo de dados.
OFFHEAP_TIERED	Entradas armazenadas em *off-heap*, ignorando a memória *on-heap*.
SWAP_SPACE	Ultrapassados os limites da memória *on-heap* e *off-heap*, configuramos um *swap space* para serem transferidas as entradas para o disco em vez de serem descartadas.
Políticas de Substituição	
Least Recently Used (LRU)	Entrada menos utilizada recentemente é substituída em primeiro lugar.
First In First Out (FIFO)	Entrada que está há mais tempo na fila, a primeira a entrar, vai ser substituída em primeiro lugar.
Modo de Rebalanceamento	
SYNC	Bloqueio de chamadas para armazenamento e distribuição na *cache* até que o reequilíbrio esteja terminado.
ASYNC	As chamadas para armazenamento e distribuição na *cache* são iniciados imediatamente, procedendo ao reequilíbrio dos dados nos nós em segundo plano.

Tabela 4 – Propriedades de Configuração de Cache Apache Ignite/GridGain.

Após execução dos intervenientes ativos, é possível utilizar a interface de linha de comandos fornecida pelo Apache Ignite. Esta fornece capacidades de monitorização e gestão da topologia mostrando estatísticas sobre os nós, *cache* e tarefas na *grid*, métricas e propriedades de configuração. Para executar, basta fazer o comando *" bash ./visorIgnite/ignitevisorcmd.sh"* e fazer *"open"* do ficheiro *"default-config.xml"* que contém os endereços IP da topologia.

A Figura 26 ilustra a execução de um processo em três servidores de cache usando a interface de linha de comandos, bem como a configuração da *cache* implementada.

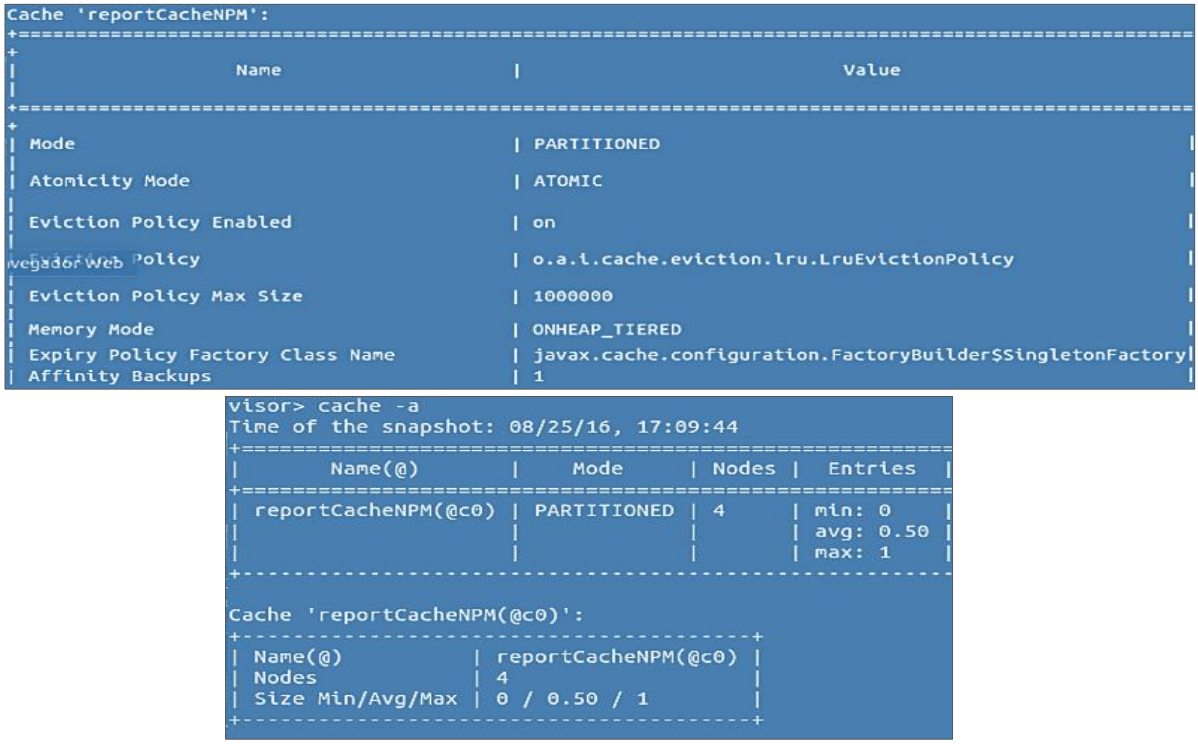


Figura 26 – Interface CLI Apache Ignite/GridGain.

3.2.7. Configuração Apache Geode/ Pivotal Gemfire

A tecnologia, Apache Geode / Pivotal Gemfire, fornece um conjunto de métodos e propriedades que ficam disponíveis com o carregamento da dependência principal no ficheiro *"pom.xml"* de cada pacote do projeto Java Maven.

A Figura 27 apresenta o esquema de execução da topologia do sistema envolvendo a comunicação e tarefas dos *servidores de cache, clientes e mediador*. *O* Pivotal Gemfire permite a utilização de mediadores para obter a localização dos servidores de *cache* e clientes, que o Apache Ignite não disponibiliza de forma explícita.

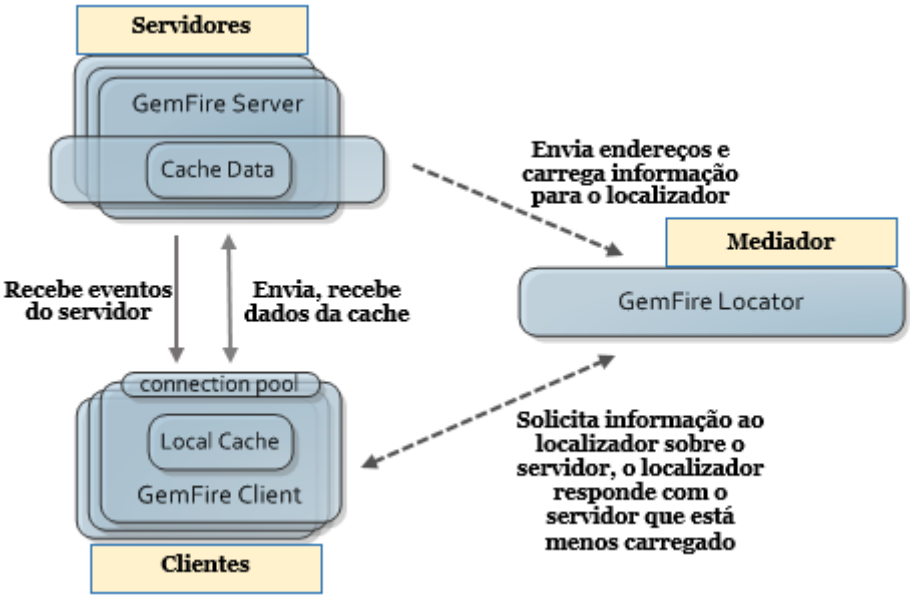


Figura 27 – Fluxo de comunicação entre Servidores de Cache, Localizador e Cliente Apache Geode/Pivotal Gemfire, adaptado de (Pivotal 2014).

Como é exemplificado programaticamente na Listagem 3 no Anexo A, o processo é lançado a partir do pacote *"LocatorLauncher" que* informa os clientes das conexões existentes aos servidores de cache ativos no sistema distribuído, fazendo posteriormente a gestão e monitorização de carga nos mesmos, enviando as solicitações de pedidos de clientes para servidores menos carregados, conduzindo desta forma a uma distribuição de entradas de forma balanceada pelos nós do *cluster*. O pacote de servidor inicia um *"ServerLauncher"* que se comunica diretamente com o localizador e permite criar e disponibilizar os servidores de *cache* para armazenamento. No servidor também definimos a topologia de cache que queremos implementar com recurso ao ficheiro configurável *"cache.xml"*. Por fim, o pacote de cliente inicia um *"ClientCache"* que se vai ligar automaticamente ao localizador existente e posteriormente aos servidores de *cache* Pivotal Gemfire ativos. É possível configurar no cliente uma *cache* local para a região existente.

De notar que a existência de comunicação entre todos os ativos deve-se à configuração no ficheiro *"config.properties",* dos endereçosIP e portas de conexão, bem como a execução do método *"getFileProperties()",* responsável por fazer a leitura das mesmas.

A configuração da topologia de *cache* em memória é fundamental para armazenar entradas nos nós dos servidores de *cache*, sob um conjunto de propriedades e características fornecidas pela própria tecnologia.

O Pivotal Gemfire fornece a capacidade de criar regiões na *cache* em modo local e em servidores de *cache*, definindo as propriedades consoante o que se pretende. A Listagem 4 no Anexo A exemplifica a criação e configuração de uma região com essas características, denominada por *"reportCache"*, com a estrutura de dados documentada na secção 3.2.3.

A região da *cache local* foi configurada programaticamente com a propriedade *"CACHING_PROXY_HEAP_LRU"* que define um estado local com capacidade de encaminhar diretamente operações para os servidores de *cache* ativos, em simultâneo, com propriedade integrada de substituição de entradas, quando a memória se encontra no limite, através da política LRU. A mesma região nos servidores de *cache* é configurada no ficheiro *"cache.xml"*, como podemos visualizar na figura abaixo. Nesta procedemos à mesma configuração efetuada nos *servidores de cache* Apache Ignite, com um tipo de região *"Partition"*, em que todas as entradas são igualmente divididas pelos nós servidor ativos, com uma cópia de redundância das mesmas, para salvaguarda de dados. A política de substituição de entradas para o caso de atingirmos o máximo de memória *on-heap*, é o LRU *(Least Recently Used),* que se caracteriza por substituir em primeiro lugar as entradas menos utilizadas recentemente. Por fim, utilizámos a política de expiração de entradas na região, o que nos permite definir um prazo de tempo de sobrevivência para as entradas.

A Tabela 5 expõe um conjunto de propriedades Gemfire para configuração da região da cache local e servidores de cache, tal como pretendemos.

Propriedade	Descrição
Modo de Região em Cache Local	
PROXY	Uma região PROXY não tem estado local e encaminha todas as operações para um servidor.
CACHING_PROXY	Uma região CACHING_PROXY tem estado local, mas também pode enviar operações para um servidor. Se o estado local não for encontrado, então a operação é enviada para o servidor e o estado local é atualizado para conter o resultado do servidor.

CACHING_PROXY_HEAP_LRU	Inclui a política de substituição de entradas LRU, a menos utilizada recentemente é substituída em primeiro lugar.
CACHING_PROXY_OVERFLOW	Move os valores das entradas para o disco quando atinge um limite máximo.
LOCAL	A região local só tem estado local e nunca envia operações para um servidor.
LOCAL_HEAP_LRU	Inclui a política de substituição de entradas LRU, a menos utilizada recentemente é substituída em primeiro lugar.
Modo de Região em Servidores de Cache	
PARTITION	Distribuição (particionamento) de dados pelo *cluster* de nós.
PARTITION_REDUNDANT	Distribuição (particionamento) de dados pelo *cluster* de nós mas ainda com uma cópia de salvaguarda dos dados.
PARTITION_PERSISTENT	Distribuição (particionamento) de dados pelo *cluster* de nós com persistência desses dados em disco.
PARTITION_REDUNDANT_PERSISTENT	Os 3 tipos de regiões particionadas acima, com vantagens de cada uma.
PARTITION_HEAP_LRU / PARTITION_REDUNDANT_HEAP_LRU	Inclui gestão de limites de *cache* (políticas de substituição) utilizando o algoritmo LRU.
REPLICATE	Os dados serão replicados por todos os nós do *cluster* (todos têm os mesmos dados).
REPLICATE_PERSISTENT	Os dados serão replicados por todos os nós do *cluster* (todos têm os mesmos dados) com persistência em disco.
REPLICATE_HEAP_LRU	Inclui gestão de limites de *cache* (políticas de subsituição) utilizando o algoritmo LRU.

Tabela 5 – Propriedades de Configuração da Região de Cache Local e Servidor de Cache Apache Geode/Pivotal Gemfire.

Tal como acontece no Apache Ignite, o Pivotal Gemfire também oferece uma interface de linha de comandos para gestão e monitorização da topologia implementada. Para aceder à mesma, basta executar o comando *"bash ./startgfsh/gfsh"* e fazer a conexão ao localizador existente com o comando *"connect –locator=endereçoIP[porta]"*, para termos acesso a toda a

topologia. A Figura 28 ilustra a topologia do sistema em memória com a execução de três servidores de *cache* e um cliente, bem como toda configuração da *cache*.

```
gfsh>list members
Member Count : 4
Coordinator  : locator1 (tiago-X555LJ(locator1:3837:locator)

  Name    | Id
--------- | ----------------------------------------------
server2   | tiago-X555LJ(server2:4026)<v2>:9473
server1   | tiago-X555LJ(server1:3926)<v1>:41788
locator1  | tiago-X555LJ(locator1:3837:locator)<v0>:48866
server3   | tiago-X555LJ(server3:4127)<v3>:53644
```

```
Name            : reportCache
Data Policy     : partition
Hosting Members : server1
                  server2
                  server3

Non-Default Attributes Shared By Hosting Members

  Type    |            Name            | Value
--------- | -------------------------- | ---------------
Region    | statistics-enabled         | true
          | entry-time-to-live.action  | DESTROY
          | entry-time-to-live.timeout | 259200
          | size                       | 4
Eviction  | eviction-algorithm         | lru-entry-count
          | eviction-action            | local-destroy
          | eviction-maximum-value     | 1000000
Partition | redundant-copies           | 1
```

Figura 28 – Interface CLI Apache Geode/Pivotal Gemfire.

Capítulo 4

Ambiente de Testes

O capítulo 4 apresenta o estudo experimental realizado para avaliar o protótipo de *software* desenvolvido para o processo de *caching* de relatórios, usando as duas tecnologias IMDG, o Apache Ignite/GridGain e Apache Geode/Pivotal Gemfire. Divide-se em dois subcapítulos, a metodologia adotada na avaliação e posteriormente a execução e análise dos testes implementados.

O ambiente de testes foi implementado e configurado na empresa Nokia Solutions and Networks Aveiro sobre a arquitetura Cliente-Servidor ilustrada na secção 3.2.5. Para isso, foi disponibilizado um conjunto de máquinas com os mesmos recursos de memória, disco e CPU, geridos por um ambiente virtualizador, e um *cluster* de dados distribuídos, agregados em tabelas horárias e diárias, armazenados no sistema de arquivos HDFS do Hadoop.

Neste processo foi utilizada uma ferramenta específica, denominada por *Yardstick Framework*, que permite desenvolver classes de testes com um conjunto amplo de parâmetros configuráveis em sistemas com arquiteturas distribuídas em memória, gerando de seguida um conjunto de resultados na forma de ficheiros e/ou gráficos, segundo um conjunto de métricas disponíveis. Esta foi integrada nos pacotes de *software* desenvolvidos e instalado em cada máquina disponibilizada, de forma a simular um ambiente de testes com um conjunto de servidores de *cache* e clientes. O objetivo da realização de testes é validar o protótipo implementado para as tecnologias selecionadas, analisando o comportamento de operações PUT e GET simultâneos aos servidores de *cache*. No final, com base nos resultados obtidos nos testes, pretende-se avaliar qual a tecnologia IMDG que oferece melhor nível de performance no processo de *caching* de relatórios, para os casos de teste específicos.

4.1. Metodologia

4.1.1. Conjunto de Dados

Para implementar um ambiente de testes válido ao protótipo funcional é indispensável o acesso a um repositório que registe milhares de entradas de indicadores de performance (KPI) da rede, por nível de objeto e período de tempo.

Esse repositório é fornecido por um sistema de arquivos *Hadoop HDFS*, que armazena agregações em tabelas horárias e diárias de hora a hora e dia a dia, através de uma *"columnar storage"*, o *Apache Parquet* do Hadoop. O acesso às agregações de dados é processado por meio de Impala, via JDBC, com recurso ao mecanismo MPP (*Processamento Paralelo Massivo*), que divide uma tarefa em várias subtarefas, para consultas SQL executadas ao HDFS, como é ilustrado na Figura 29.

Podíamos ter optado por Hive, que utiliza mecanismos de Map Reduce, mas este consome recursos intensivos ao nível de I/O, o que tem custos muito elevados ao nível de performance de execução, ao invés do Impala, que faz mais uso da memória para o processamento.

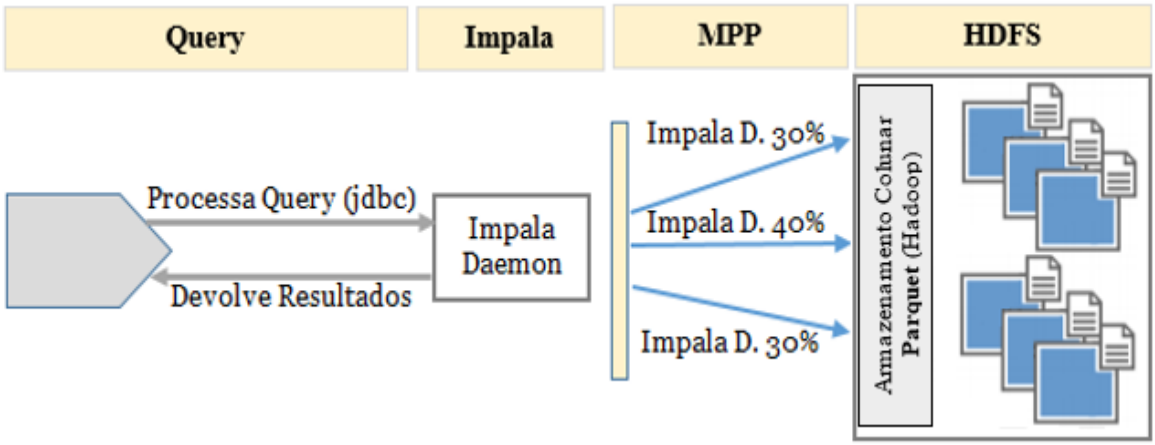


Figura 29 – Processamento de queries, via Impala.

Sendo que a quantidade de dados não é um problema, o que pretendemos fazer nos diversos testes executados, é processar centenas de consultas, via JDBC Impala, que permitam gerar relatórios de rede e armazená-los nos servidores de *cache* IMDG, devolvendo-os sempre que sejam solicitados.

Com auxílio da classe java *"generateXMLQuery.java"* foi desenvolvido um método que permite gerar um ficheiro XML com um conjunto dimensionável de consultas randomizadas. A Figura 30 ilustra um pequeno exemplo de consultas geradas através da classe mencionada. Cada consulta representa o armazenamento de um relatório de rede nos servidores de *cache*, em que os valores dos parâmetros *"aggregationLevel"*, *"id_object"*, *"means"*, *"period_start_time"* e *"period_end_time"* vão formar a chave do relatório, (p.e: *<PLMN-PLMN/MRBTS-7#02-07-2016 05:53:26#02-07-2016 06:53:26#day#lcelld>*), e as entradas obtidas da consulta SQL a lista de valores (ObjectValue), sobre a estrutura de dados pares Chave/Valor definida na secção 3.2.3. De notar que o parâmetro *"period_start_time"* tem uma entrada que vai ser substituída pela data e hora do sistema, para que seja possível utilizar as mesmas consultas para gerar relatórios diferentes, em períodos de tempo também eles diferentes.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<queries>
    <query aggregationLevel="Hour" id_object="PLMN-PLMN/MRBTS-8/LNBTS-1" means="lisho"
           period_end_time="endParam" period_start_time="startParam">
        <sql><![CDATA[select * from noklteraw.noklte_ps_lmac_lnbts_hour a where
            a.period_start_time_p>='startParam' and a.period_start_time_p< 'endParam'
            and a.lnbts_id='PLMN-PLMN/MRBTS-8/LNBTS-1']]></sql>
    </query>
    <query aggregationLevel="Day" id_object="PLMN-PLMN/MRBTS-7" means="lianbho"
           period_end_time="endParam" period_start_time="startParam">
        <sql><![CDATA[select * from noklteraw.noklte_ps_lhorlf_mrbts_day a where
            a.period_start_time_p>='startParam' and a.period_start_time_p< 'endParam'
            and a.mrbts_id='PLMN-PLMN/MRBTS-7']]></sql>
    </query>
    <query aggregationLevel="Day" id_object="PLMN-PLMN/MRBTS-24/LNBTS-1" means="lcellt"
           period_end_time="endParam" period_start_time="startParam">
        <sql><![CDATA[select * from noklteraw.noklte_ps_lianbho_lnbts_day a where
            a.period_start_time_p>='startParam' and a.period_start_time_p< 'endParam'
            and a.lnbts_id='PLMN-PLMN/MRBTS-24/LNBTS-1']]></sql>
    </query>
</queries>
```

Figura 30 – Ficheiro XML de Queries para Caching Report.

4.1.2. Carga de Trabalho

A carga de trabalho ou *workload* realizado no ambiente de testes passa por simular operações de PUT e GET simultâneos aos servidores de *cache*, recolhendo resultados de métricas

para análise comportamental da *cache*. A Figura 31 representa o fluxo de teste para as classes implementadas, *"GeodePUTGETXMLQueryBenchmark().java"* e *"IgnitePUTGETXMLQueryBenchmark().java",* respetivamente ao Pivotal Gemfire e Apache Ignite.

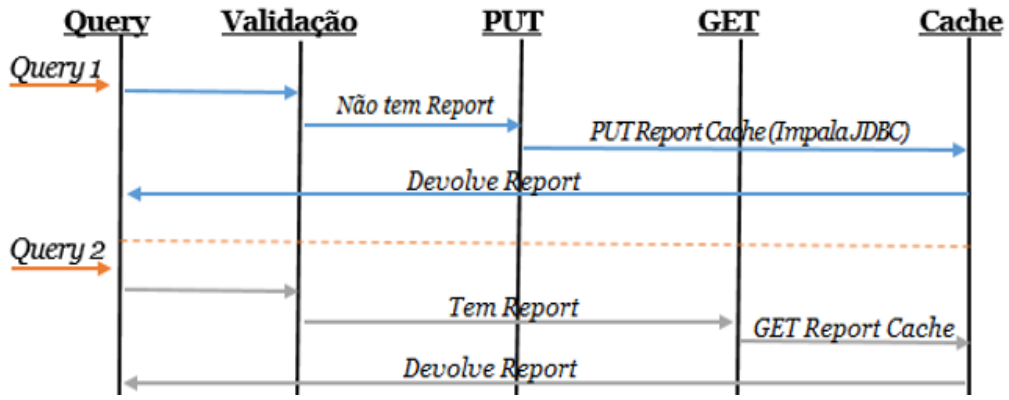


Figura 31 – Processo PUT GET Benchmark.

A ferramenta Yardstick oferece um conjunto de métodos para a implementação da classe de teste, tais como: o *"setUp (…) "* para executar um certo tipo de tarefas antes da execução do teste; e *"test (…) "* para realizar o teste propriamente dito, como é ilustrado programaticamente na Listagem 5 no Apêndice A. O método de "setUp(…)", utiliza a classe "LoadXMLQueries.java" para fazer a leitura de um conjunto extenso de consultas e parâmetros provenientes do ficheiro XML, "queries.xml". Posteriormente procedemos à decomposição e adição numa estrutura *Map* em memória das chaves e consultas associadas aos vários relatórios de rede. No método *"test(…)"*, procedemos à seleção aleatória, na estrutura *Map*, de uma chave do relatório (p.e: <*PLMN-PLMN/MRBTS-7#startParam#endParam#day#lcelld*>) à qual está associado o valor correspondente à consulta SQL. De seguida obtemos a data e hora do sistema e fazemos a respetiva formatação de *"startParam"* e *"endParam"*, assente na seguinte lógica:

- Se a chave for de um relatório agregado à hora, transforma-se a hora, ou seja, se forem "09:20:00", assumimos como "09:00:00" e se forem "09:31:00", assumimos como "10:00:00", e subtraímos ainda uma hora ao *"endParam"* e outra ao *"startParam"*, mantendo a data atual do sistema.

- Se a chave for de um relatório agregado ao dia, transforma-se a hora da mesma forma e à data atual do sistema, o *"endParam"*, subtraímos um dia, obtendo o *"startParam"*.

Estas formatações são executadas para conseguir obter os relatórios agregados, da hora e dia anteriores e nunca os atuais, garantindo que as consultas sejam sempre processadas e que tenham sempre dados para gerar o relatório. Caso contrário, não existiriam dados, pois as agregações só são processadas de hora a hora e/ou dia a dia.

Depois de transformar o *"startParam"* e o *"endParam"* num *timestamp* válido na chave e na consulta SQL, podemos fazer o processamento de relatórios de rede.

Numa primeira fase verificamos a existência da chave do relatório na *cache*, através do método *"getKeyReport(key)"*, e caso exista devolvemos o relatório. Caso não exista, iniciamos a conexão, via JDBC Impala, e através do método *"storeResultsQueries(key, query, conn)"*, geramos o relatório e procedemos ao seu armazenamento *(PUT)* sobre a estrutura de dados pares Chave/Valor ilustrada na secção 3.2.3, na *cache* dos servidores ativos. Posteriormente, ao ser requisitada a mesma chave, será devolvido *(GET)* diretamente da cache.

Este procedimento é bastante simples e vai servir para fazer os testes de carga repetitivos, em vários períodos de tempo, por forma a analisar mais concretamente o comportamento da *cache* para as tecnologias Apache Ignite e Pivotal Gemfire.

4.1.3. Métricas de Avaliação

A ferramenta de testes, Yardstick Framework, coleta métricas de desempenho durante a execução dos testes, organizando-se em grupos como: Taxa de transferência, Latência, CPU, Processos, Sistema, IO, Swap e Memória. A Tabela 6 refere essas métricas e respetivas descrições, que vão servir para analisar os resultados referentes aos testes efetuados (Alexandre 2009).

#	Métrica	Descrição
ThroughputLatencyProbe – Sonda de medição de taxa de transferência e latência.		
Outros	Operações /seg	Número de operações executadas por segundo.
	Latência /seg	Período de tempo na execução de operações.

VmStatProbe – Sonda de medição de informação sobre Processos, Memória, Swap, IO, Sistema, CPU.		
Processos	Processos em espera	Número de processos em execução aguardando acesso à CPU.
	Processos em bloqueio	Número de processos bloqueados.
Memória	Memória Usada /KB	Quantidade de memória virtual utilizada.
	Memória Livre /KB	Quantidade de memória virtual disponível.
	Memória Buffered /KB	Quantidade de memória virtual alocada para *buffer* ou região de armazenamento temporário.
	Memória Cache /KB	Quantidade de memória virtual alocada como *cache*.
Swap	Swapped a partir do disco /seg	*Swap in*, quantidade de dados transferidos do disco para a memória.
	Swapped para o disco /seg	*Swap out*, quantidade de dados transferidos, da memória para o disco.
IO	Blocos IO Recebidos /seg	Blocos IO (Leitura/Escrita) recebidos por dispositivos de blocos[8], recorrendo à utilização do disco.
	Blocos IO Enviados /seg	Blocos IO (Leitura/Escrita) enviados por dispositivos de bloco, recorrendo à utilização do disco.
Sistema	Interrupções de Sistema /seg	Número de interrupções de sistema.
	Context Switch de Sistema /seg	Número de *context switches*[9] de sistema.
CPU	CPU User /%	Percentagem de tempo consumido pela CPU na execução de processos de utilizador (não-Kernel), tais como: bases de dados e execução de programas.
	CPU System /%	Percentagem consumida pela CPU na execução de processos Kernel, tais como: memória e IO.
	CPU Idle /%	Percentagem de tempo de inatividade do CPU.
	CPU Wait /%	Percentagem de tempo de CPU ocupado por processos em estado *waiting*.

Tabela 6 – Métricas Avaliadas pela ferramenta Yardstick Framework.

8 Dispositivos que podem enviar/transmitir dados em blocos de tamanho fixo, como é o exemplo do disco rígido, pendrives e cdrom.
9 Procedimento que o sistema segue, no processamento paralelo, para mudar de uma tarefa ou processo para outro, assegurando que estas não entram em conflito.

4.1.4. Casos de Teste

Os pacotes de *software* implementados, com a framework Yardstick integrada, foram configurados e instalados no ambiente de testes sobre a arquitetura Cliente-Servidor enunciada na secção 3.2.5. Para isto, utilizámos quatro máquinas, três como *servidores de cache* e uma como *cliente* ou *driver* de execução dos testes. As respetivas máquinas com os endereçosIP, 10.92.116.160, 10.92.116.161, 10.92.116.162 e 10.92.116.168 têm os seguintes recursos físicos:

- **CPU** – 2CPU Xeon X5670, 2.93GHz c/6 cores.
- Memória RAM – 198 GB.
- **Disco** – 2x900GB HDD.
- **SO** – Linux Rhel (Red Hat) 6.7.
- **Outros** – Java Oracle JDK 1.8, SSH Cliente/Servidor.

O plano de testes idealizado passa pela realização de dois casos de teste distintos. O primeiro caso de teste divide-se em quatro testes configurados para executar durante 30 minutos, 1 hora, 2 horas e 3 dias. A configuração dos servidores de *cache* para os testes executados assenta num modo de *cache* particionado com uma cópia dos dados, utilizando somente a memória OnHeap-Tiered e com uma política de substituição de entradas LRU.

O segundo caso de teste executa um quinto teste durante 2 horas, em que a configuração dos servidores de *cache* é exatamente igual à utilizada no primeiro caso de teste com a particularidade de incluir a aplicação de uma política de expiração de entradas na *cache*, num prazo limite de 1 hora, permitindo validar o comportamento da mesma quando procedemos à remoção de entradas.

A Figura 32 ilustra o plano de testes executado para cada tecnologia IMDG, o Apache Ignite e Pivotal Gemfire respetivamente.

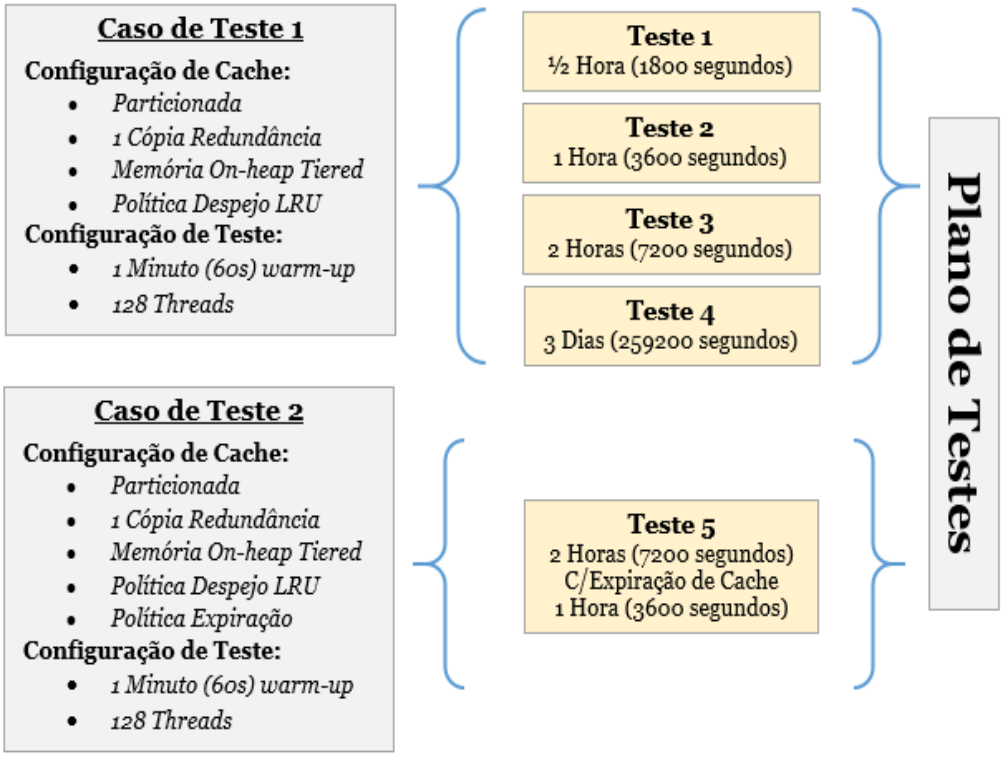


Figura 32 – Plano de Testes para Ignite e Gemfire.

4.2. Resultados

Nesta secção são apresentados os resultados obtidos pela framework Yardstick para o plano de testes mencionado na secção acima. Para isto, procedeu-se à seleção de um conjunto de métricas enquadradas nos grupos da taxa de transferência e latência, consumo de memória e CPU e dispositivos de bloco IO.

A tabela 7 representa o resumo dos valores médios aproximados, derivado dos resultados obtidos nos testes, tendo em consideração o tempo de execução dos mesmos, para as métricas selecionadas, nos dois casos de teste, para as tecnologias Apache Ignite e Pivotal Gemfire respetivamente. De notar que foi desenvolvida uma página web simples, com o intuito de concatenar e permitir uma melhor visualização de todos os resultados relevantes, inclusive os valores das restantes métricas não avaliadas.

Métricas Avaliadas		Caso de Teste 1								Caso de Teste 2	
		Teste 1 30 Minutos (1800s); 128 threads; 1 Minuto (60s) warm-up;		**Teste 2** 1 Hora (3600s); 128 threads; 1 Minuto (60s) warm-up;		**Teste 3** 2 Horas (7200s); 128 threads; 1 Minuto (60s) warm-up;		**Teste 4** 3 Dias (259200s); 128 threads; 1 Minuto (60s) warm-up;		**Teste 5 c/Expiração** 2 Horas (7200s); 128 threads; 1 Minuto (60s) warm-up;	
IMDG		**Gemfire**	**Ignite**	**Gemfire**	**Ignite**	**Gemfire**	**Ignite**	**Gemfire**	**Ignite**	**Gemfire**	**Ignite**
#ThroughputLatency (Valores médios aproximados)											
Outros	Operações/seg	118,151	37,493	126,219	36,895	129,400	39,408	131,050	33,434	100,948	35,500
	Delta [10]	215%		242%		228%		292%		184%	
	Latência/seg	1,33	0.99	0,99	0,89	0,81	0,73	0,53	1,41	3,55	1,01
#VmStat (Valores médios aproximados)											
Memória	Memória Livre	41,7Gb	41,5Gb	38,1Gb	38,0Gb	38,0Gb	38,0Gb	39,7Gb	37,7Gb	39,7Gb	39,2Gb
	Memória Cache	0,97Gb	0,98Gb	2,7Gb	2,7Gb	2,7Gb	2,7Gb	1,3GB	2,4Gb	1,1Gb	1,5Gb
	Memória Buffer	165Mb	165Mb	178Mb	175Mb	176Mb	177Mb	162Mb	175Mb	166Mb	166Mb
IO	Blocos IO Enviados/seg	22,95	21,93	13,13	11,71	14,28	11,06	16,55	71,87	60,27	57,04
	Blocos IO Recebidos/seg	0,01	0,01	0,00	0,00	0,00	0,00	0,12	0,06	0,10	0,01
CPU	CPU User /%	68,96%	64,95%	73,53%	65,43%	76,75%	68,69%	77,10%	59,12%	59,75%	63,56%
	CPU System /%	0,35%	6,23%	0,38%	6,12%	0,40%	6,50%	0,37%	5,90%	0,42%	6,10%
	CPU Idle /%	30,57%	28,75%	25,97%	28,30%	22,76%	24,71%	22,41%	34,88%	39,75%	30,19%

Tabela 7 – Resultados numéricos dos Casos de Teste 1 e 2 .

10 Diferença percentual entre valores Ignite e Gemfire (%). **Fórmula**: (B-A) /A*100.

4.2.1. Taxa de Transferência e Latência

A Figura 33 ilustra os resultados gráficos para o caso de teste 1. Este divide-se em quatro testes similares, onde apenas é alterado o tempo de execução, numa análise ao número de *operações* e respetiva *latência* para Apache Ignite e Pivotal Gemfire.

Como se pode verificar em cada teste, as operações PUT são deveras mais demoradas do que as operações GET, o que se confirma ao verificar a latência correspondente. Isto porque requerem acesso e pesquisa à base de dados externa e armazenamento do relatório em *cache*, sendo desta forma expectável o aumento no tempo de processamento deste tipo de operações. Por sua vez, as operações GET são quase instantâneas porque apenas verificam a existência do relatório de rede na cache e procedem à respetiva devolução.

No que se refere à análise das tecnologias IMDG estudadas verifica-se que o Pivotal Gemfire processa mais operações por segundo do que o Apache Ignite:

- **Teste 1** de 30 minutos, temos aproximadamente 118,151 para 37,493 operações/seg, correspondendo a 215% de diferença.
- **Teste 2** de 1 hora, temos 126,219 para 36,895 operações/seg, correspondendo a 242% de diferença.
- **Teste 3** de 2 horas, temos 129,400 para 39,408 operações/seg, correspondendo a 228% de diferença.
- **Teste 4** de 3 dias, temos 131,050 para 33,434 operações/seg, correspondendo a 292% de diferença.

Esta diferença deve-se ao facto do Pivotal Gemfire ser bastante mais rápido a processar operações GET. Contudo, nas operações PUT verifica-se que o Pivotal Gemfire tem maior latência do que o Apache Ignite para os testes 1,2 e 3, exceto no teste 4, o que pode ser explicado pela estabilidade na execução de operações aquando um período mais alargado de teste ou com fatores de comunicação com o *cluster* de dados externo.

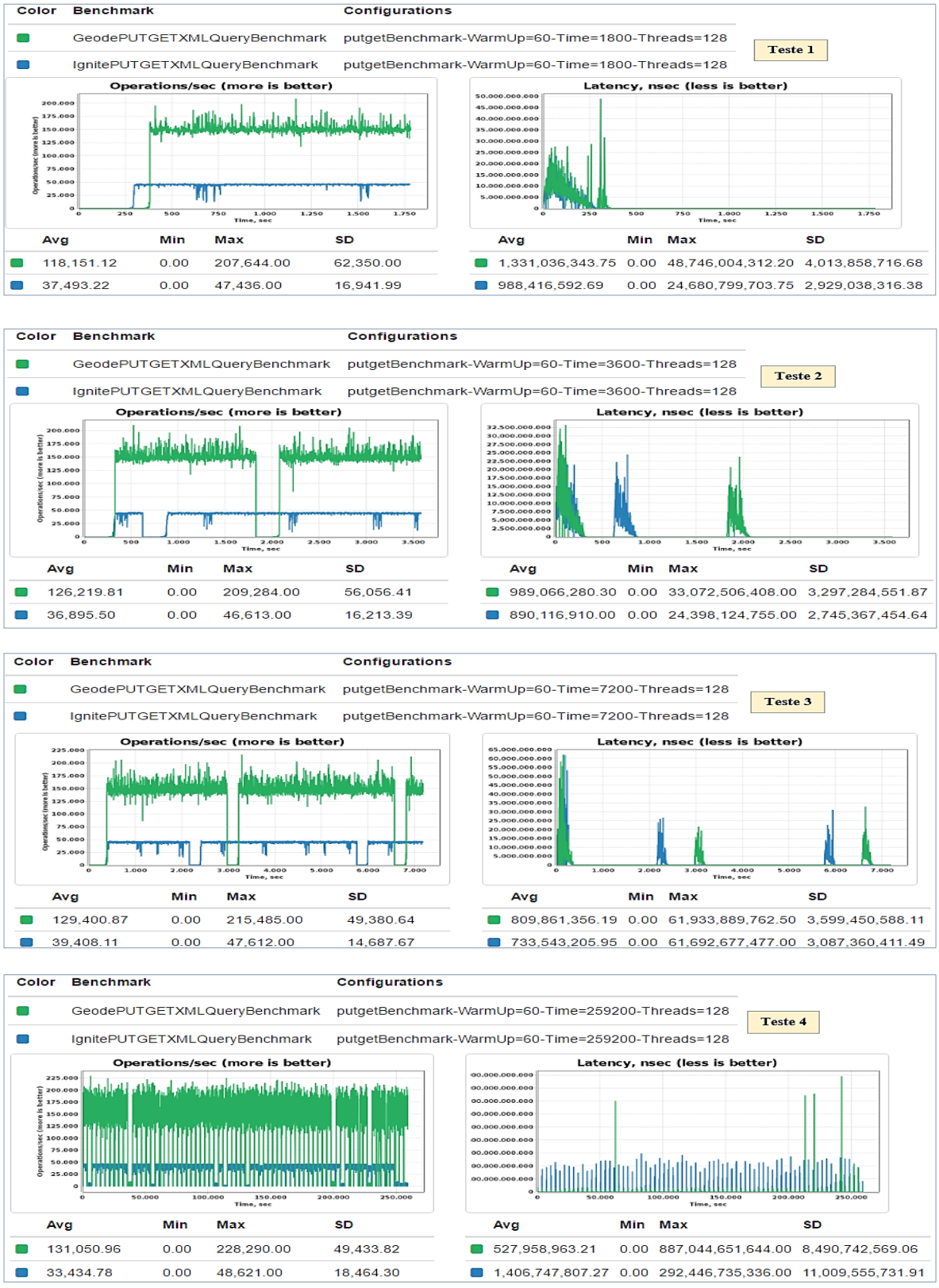


Figura 33 – Resultados gráficos “Taxa de Transferência e Latência”, do Caso de Teste 1.

A Figura 34 ilustra os resultados gráficos para o *caso de teste 2*, na execução do teste 5 de 2 horas, que pretende validar o conceito de política de expiração no período de tempo de 1 hora, analisando o número de operações e respetiva latência para Apache Ignite e Pivotal Gemfire.

Com a política de expiração ativa verificamos que tanto o Pivotal Gemfire como o Apache Ignite têm um decréscimo considerável no número de operações/seg executadas e aumento de latência em comparação com o teste 3 na mesma medida de tempo.

- **Teste 5** de 2 horas, temos aproximadamente 100,948 para 35,500 operações/seg, correspondendo a 184% de diferença.

Como podemos visualizar no teste 5, o Apache Ignite aos 3600 segundos procede à respetiva remoção de entradas e execução de operações PUT. Apesar da política de expiração estar ativa para o Pivotal Gemfire, este faz uma gestão de remoção completamente diferente do Apache Ignite. Isto é, enquanto o Apache Ignite determina o período de expiração a partir do momento em que existem entradas na cache, o Pivotal Gemfire define esse período pelo último acesso à entrada na *cache*.

Sabendo que o teste executado é contínuo e as operações de PUT e GET estão a ser processadas simultaneamente, a política de expiração para Pivotal Gemfire não é passível de visualizar, apesar de estar ativa. A solução encontrada para validar a política de expiração para Pivotal Gemfire passou por parar o teste, não executando qualquer operação durante o período de tempo de 1 hora, verificando posteriormente a existência das entradas na cache, ao que se verificou que a remoção das entradas é feita naturalmente.

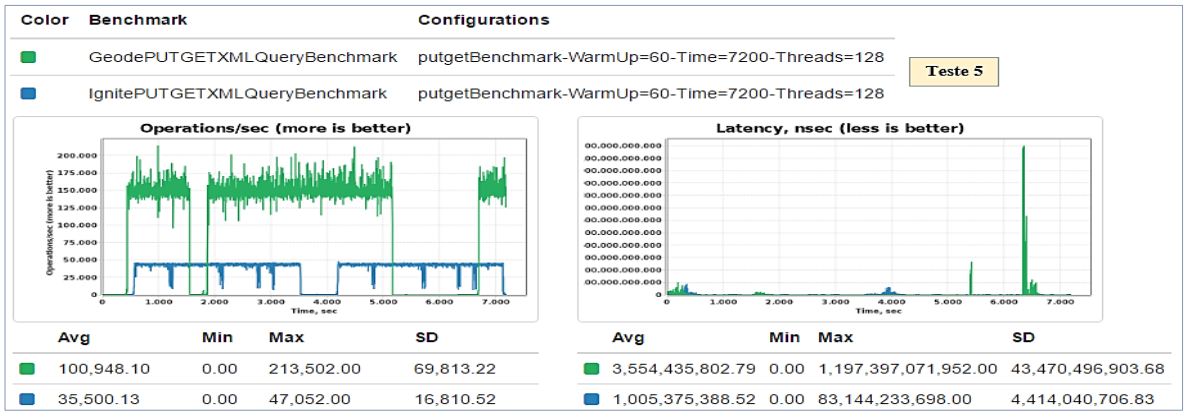


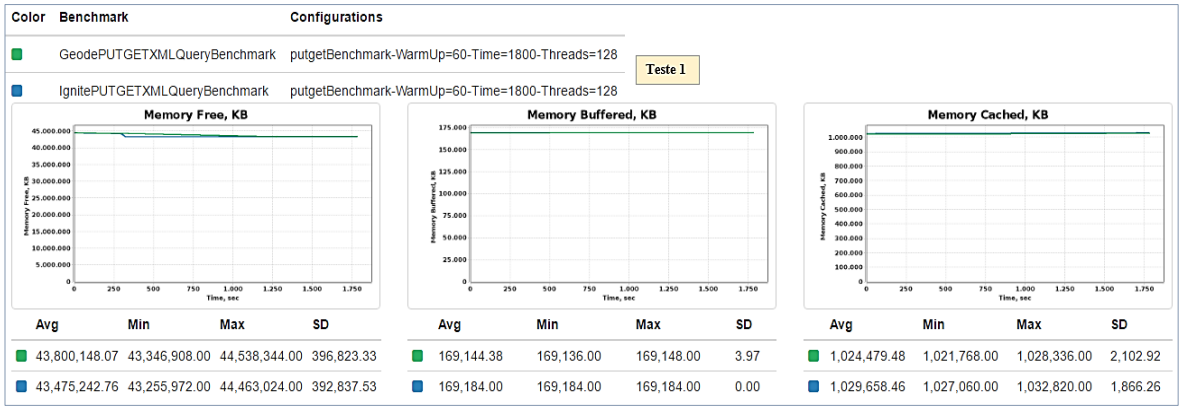
Figura 34 – Resultados gráficos "Taxa de Transferência e Latência", do Caso de Teste 2.

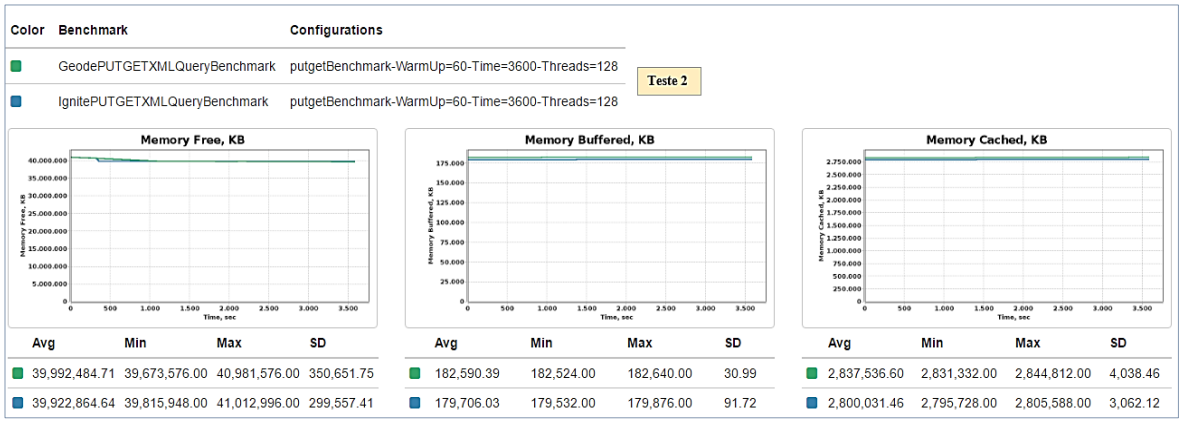
4.2.2. Consumo de Memória

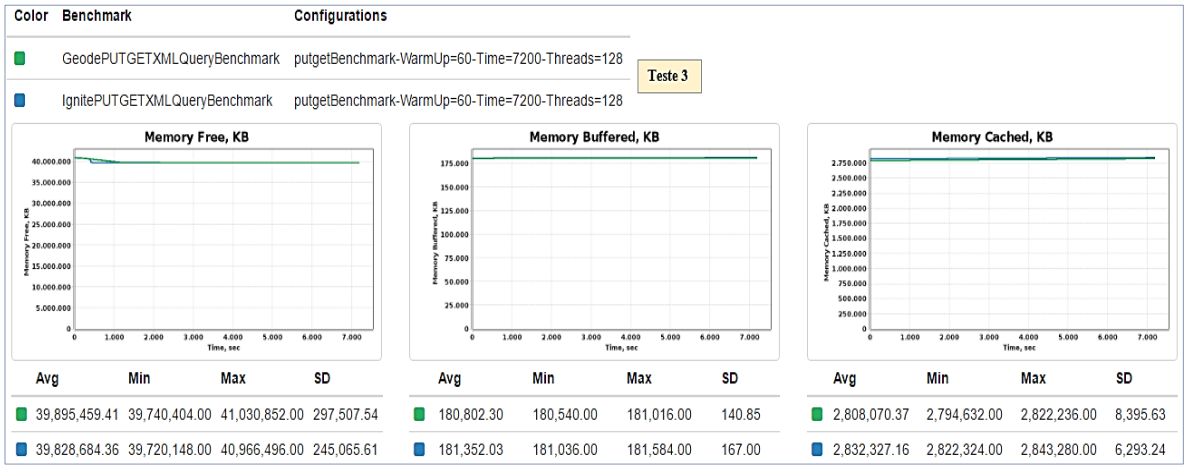
A Figura 35 ilustra os resultados gráficos do consumo de memória para os casos de teste 1 e 2 referentes às tecnologias Apache Ignite e Pivotal Gemfire.

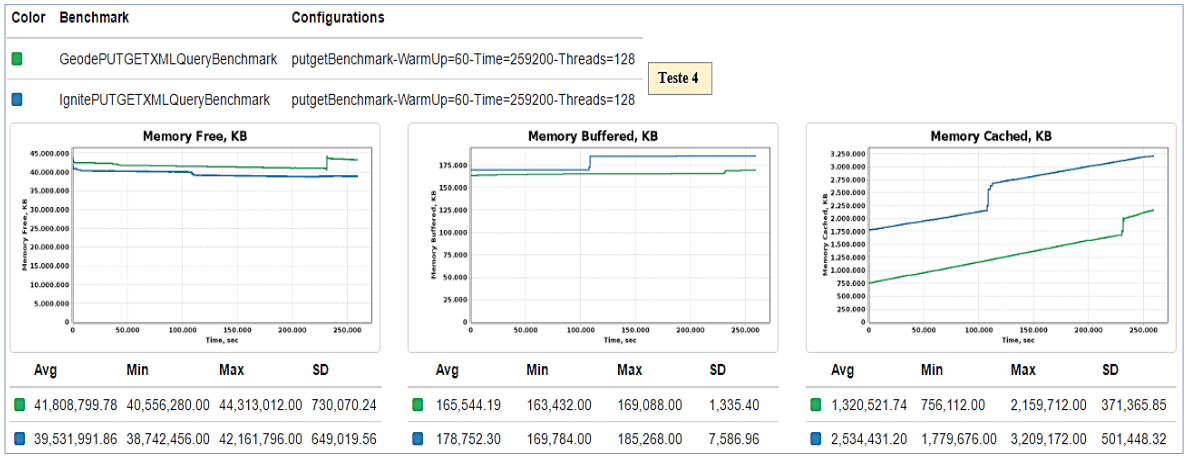
No que se refere à alocação de memória *buffer* e *cache* necessárias para execução dos testes a diferença não é substancial, o que quer dizer que o consumo é bastante similar entre as duas tecnologias. Contudo no teste 4 de 3 dias existe alguma discrepância nos resultados, onde o Pivotal Gemfire necessita de alocar 1,3Gb para *cache* e 162 Mb para *buffer* e o Apache Ignite 2,4Gb para *cache* e 175Mb para *buffer*, ou seja, o Apache Ignite precisa de mais 1,1Gb de cache e 13Mb de buffer do que o Pivotal Gemfire. Apesar da similaridade dos resultados para testes de curto prazo, tudo indica que em testes a longo prazo o Pivotal Gemfire faz uma melhor gestão de memória do que o Apache Ignite.

Os resultados obtidos para a memória livre em todos os testes ascendem naturalmente a Gb, pois a memória não é totalmente utilizada, o que requeria um maior número de entradas na cache, bem como a execução de testes de maior período de tempo.









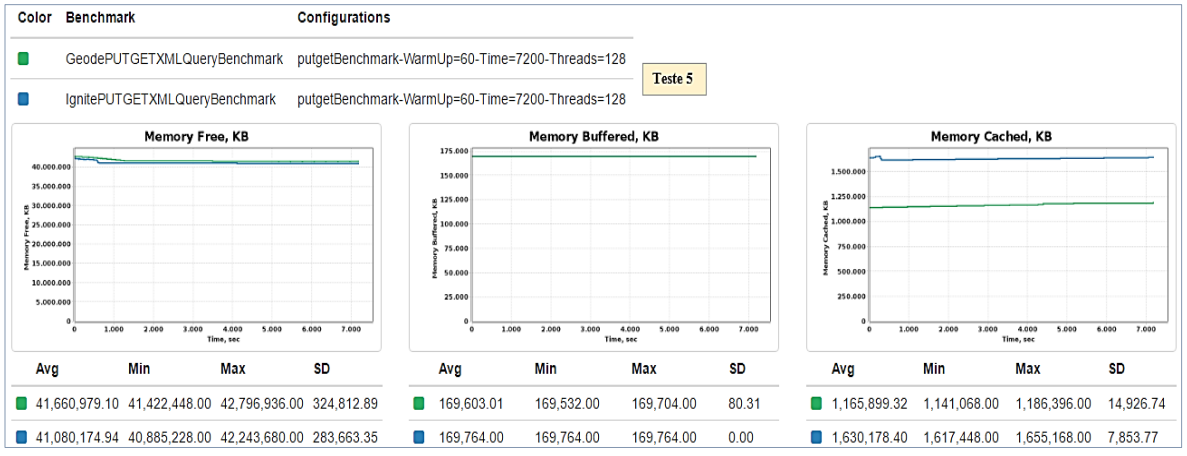


Figura 35 – Resultados gráficos “Consumo de Memória”, dos Casos de Teste 1 e 2.

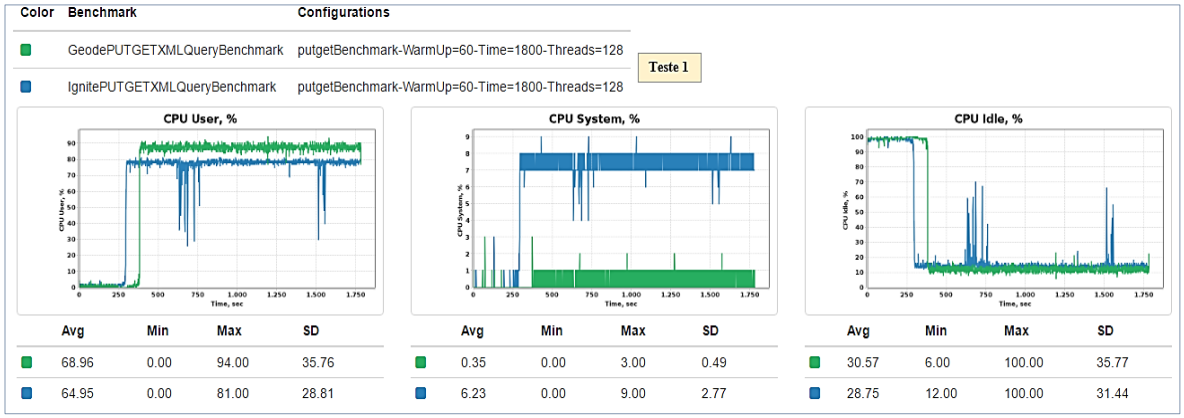
4.2.3. Consumo de CPU

A Figura 36 ilustra os resultados gráficos do consumo de CPU para os casos de teste 1 e 2 referentes às tecnologias Apache Ignite e Pivotal Gemfire.

Numa análise geral aos cinco testes executados, no que corresponde ao consumo de *CPU User*, ou seja, os processos de utilizador, o Pivotal Gemfire em média consome por cada teste cerca de 71,21% para os 64,3% consumidos pelo Apache Ignite. Relativamente ao consumo de *CPU System*, em processos de *kernel* (p.e: alocação de memória, sistema), o Pivotal Gemfire consome em média por cada teste executado cerca de 0,38% para os 6,17% consumidos pelo Apache Ignite.

Para a percentagem de *CPU Idle*, o Pivotal Gemfire em média por cada teste tem uma percentagem de inatividade de 28,3% para 29,36% do Apache Ignite.

Apesar do Pivotal Gemfire consumir maior percentagem de *CPU User* na utilização do próprio *software* IMDG e acesso ao cluster de dados externo, o Apache Ignite necessita de maior percentagem de *CPU System* na alocação de memória e operações de sistema, o que vai de encontro com os resultados obtidos.



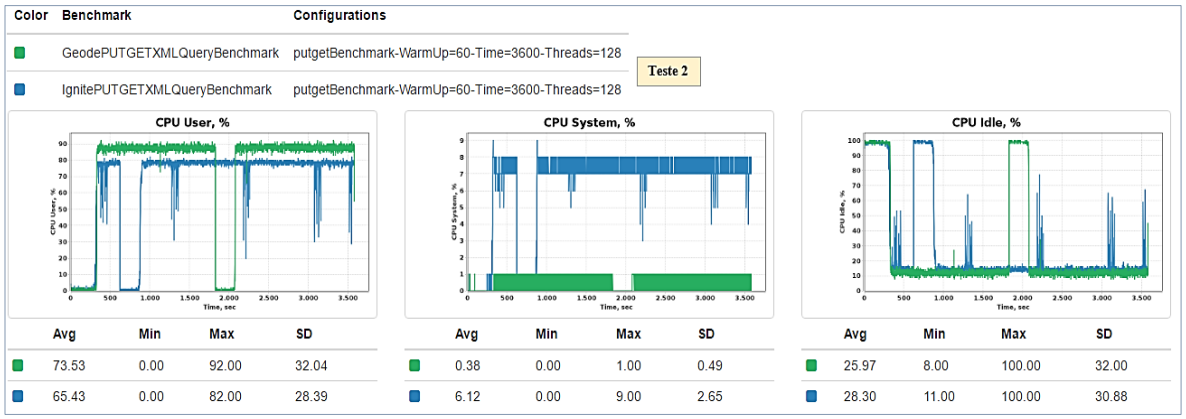


Figura 36 – Resultados gráficos “Consumo de CPU”, dos Casos de Teste 1 e 2.

4.2.4. Comportamento de Operações IO (Leitura/Escrita)

A Figura 37 ilustra os resultados gráficos do comportamento de operações IO para os casos de teste 1 e 2 referentes às tecnologias Apache Ignite e Pivotal Gemfire. As operações IO, no sistema IMDG configurado, referem-se ao volume de leituras e escritas por segundo em blocos fixos, utilizando recursos físicos (**ex:** Disco). No entanto, como estamos a falar de tecnologias em memória, a utilização deste tipo de operações é reduzida. Como podemos verificar nos testes executados, os *blocos IO recebidos* para Pivotal Gemfire e Apache Ignite são praticamente nulos, isto porque não existem operações de escrita em disco, pois estamos perante tecnologias que fazem operações de PUT e GET diretamente da *cache* em memória. Algo que poderia influenciar o comportamento do IO seria a configuração de *backup e swap*, mas o *backup* utilizado na *cache* em nada altera o comportamento deste tipo de operações, pois esta é feita totalmente em memória, e o espaço *swap* não foi configurado neste sistema o que reduz ainda mais a possibilidade de utilizar o disco nas respetivas operações de leitura e escrita.

Pelo lado inverso, o comportamento nas operações de *blocos IO enviados* não é similar, isto porque procedemos a algum processamento paralelo recorrendo ao disco em ações de leitura de chaves randomizadas e respetivas consultas do relatório de rede, bem como configurações de ligação ao *cluster* de dados e envio de consultas para execução.

Numa análise geral aos testes executados pode-se concluir que o Pivotal Gemfire e o Apache Ignite obtêm valores para operações de blocos IO enviados muito idênticos, na ordem dos 11 a 72 blocos por segundo consoante o prazo de duração do teste.

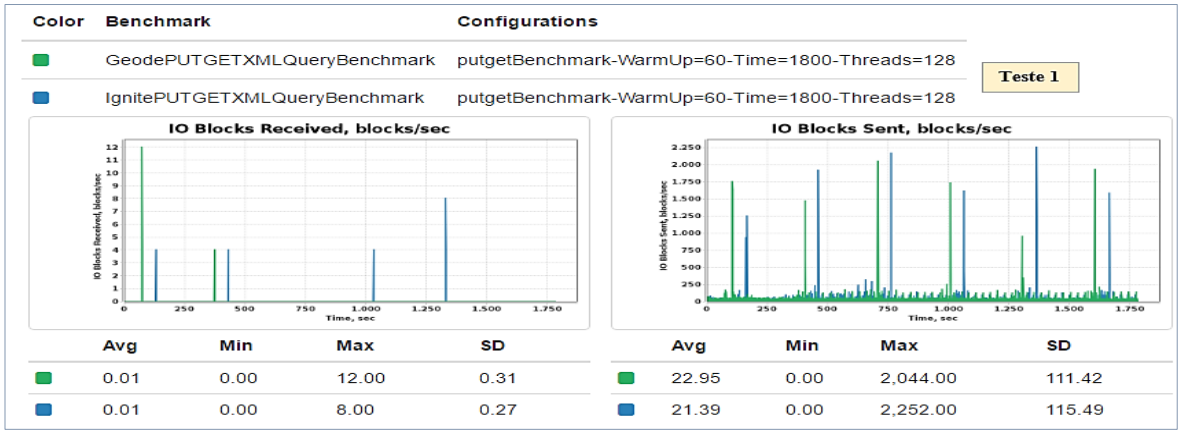


Figura 37 – Resultados gráficos “Operações IO”, dos Casos de Teste 1 e 2.

4.3. Discussão

Numa análise geral ao cenário de avaliação foram testadas duas tecnologias IMDG, Pivotal Gemfire e Apache Ignite, com recurso a uma ferramenta de testes para sistemas distribuídos em memória, denominada por Yardstick Framework. A taxa de transferência e latência nas operações, o consumo de memória e CPU e o comportamento de operações de bloco IO, foram avaliadas no decorrer de dois casos de teste num conjunto de cinco testes com períodos de tempo de execução diferentes, onde ainda houve lugar para validar a política de expiração de entradas na *cache*. É preciso ter em consideração que os resultados obtidos nos testes executados poderão em alguns casos ter sofrido anomalias derivadas da estabilidade do *cluster* de dados externo e tempo de resposta na execução de operações PUT, isto porque o *cluster* de dados utilizado está em fase experimental e ainda está ser alvo de afinações.

Com base nos resultados conseguidos, neste cenário de testes específico, podemos determinar um conjunto de conclusões relativas ao comportamento do sistema ao nível de performance, para as duas tecnologias estudadas.

Taxa de Transferência e Latência

- O Gemfire processa mais operações por segundo do que o Ignite, com diferenças entre 180% - 292%.
- Testes de curto prazo de tempo, entre *30 minutos - 2 horas*:
    - *Gemfire:* (**mais**) latência em operações PUT, (**menos**) latência em operações GET.
    - *Ignite:* (**menos**) latência em operações PUT, (**mais**) latência em operações GET.
- Testes de longo prazo de tempo, de *3 dias*:
    - *Gemfire:* (**menos**) latência em operações PUT, (**menos**) latência em operações GET.
    - *Ignite:* (**mais**) latência em operações PUT, (**mais**) latência em operações GET.

Consumo de Memória

- Testes de curto prazo de tempo, entre *30 minutos - 2 horas*:
  - *Gemfire e Ignite:* similaridade na memória alocada para *buffer* e *cache*.
- Testes de longo prazo de tempo, de *3 dias*:
  - *Gemfire:* (**menos**) memória alocada para *buffer* e *cache*, (**menos**) memória utilizada, (**mais**) memória disponível.
  - *Ignite:* (**mais**) memória alocada para *buffer* e *cache*, (**mais**) memória utilizada, (**menos**) memória disponível.

Consumo de CPU

- Testes de curto e longo prazo de tempo, entre *30 minutos – 3 dias*:
  - *Gemfire:* (**mais**) consumo de CPU User, (**menos**) consumo de CPU System, percentagem de CPU Idle estável para prevenir possíveis picos de consumo.
  - *Ignite:* (**menos**) consumo de CPU User, (**mais**) consumo de CPU System, percentagem de CPU Idle estável para prevenir possíveis picos de consumo.

Comportamento de Operações IO (Leitura/Escrita)

- Testes de curto e longo prazo de tempo, entre *30 minutos – 3 dias*:
  - *Gemfire e Ignite:* similaridade nos blocos IO enviados/recebidos por segundo, com valores reduzidos de leituras e escritas no disco, pois estamos a trabalhar com tecnologias em memória.

Política de Expiração de Entradas

- Teste de *2 horas* com expiração de entradas após *1 hora*:
  - *Gemfire:* expiração efetuada após 1 hora, sem acessos PUT e GET às entradas.
  - *Ignite:* expiração efetuada após 1 hora independentemente de existirem acessos PUT e GET às entradas.

As conclusões obtidas representam de certa forma uma vantagem para a tecnologia Pivotal Gemfire, neste cenário de avaliação. Para além de processar um maior número de operações por segundo com níveis de latência reduzidos, ainda consegue fazer uma melhor gestão da memória disponível, o que vai de encontro com o baixo processamento de CPU System utilizado.

Os resultados atingidos pelo Pivotal Gemfire poderão estar ligados à forma como a plataforma faz a gestão interna de entradas nos servidores de *cache*, e algo que pode explicar este facto é a utilização de uma camada mediadora ou localizadores que complementam a sua arquitetura. Esta tem um papel fulcral na comunicação entre todos os ativos disponíveis, bem como na distribuição balanceada de entradas na cache dos servidores.

Este estudo poderá ter um impacto na forma como a organização, Nokia Solutions Aveiro, pretende prosseguir com a possível remodelação da plataforma *Nokia Performance Manager* no processo de *caching* de relatórios, bem como dar origem a possíveis estudos por parte de outras organizações no mercado, sabendo de antemão que as tecnologias IMDG podem vir a ser muito relevantes no armazenamento e processamento de grandes quantidades de informação no futuro.

Capítulo 5

Conclusões

Uma *"In-Memory Datagrid"* é uma estrutura de dados que reside inteiramente na memória, e é distribuída num conjunto de servidores, tornando prático o armazenamento de *terabytes* de dados, evitando a necessidade de utilização de dispositivos eletromecânicos, como discos rígidos. São adequadas para lidar com os *"V's"* do Big Data, oferecendo alta performance, escalabilidade, disponibilidade e uma estrutura de dados chave/valor flexível.

O estudo baseou-se na utilização deste conceito na atividade de processar analiticamente dados em aplicações de monitorização de performance de redes usando a ferramenta Nokia Performance Manager, num processo específico de armazenamento de relatórios de rede.

Com base neste desafio, foram analisados um conjunto de produtos IMDG, na qual procedemos à seleção do Apache Geode / Pivotal Gemfire e Apache Ignite / Grid Gain. Estes, por terem uma grande relevância no mercado das IMDGs, oferecerem plataformas *open source* e recursos idênticos ao nível de características de uma *datagrid* com suporte para *Streaming* e *Hadoop* para futuras integrações, serviram como casos de estudo para a tese de dissertação.

Foi desenvolvido um protótipo de *software* para cada IMDG, com base numa arquitetura distribuída real e uma estrutura de dados adaptada à ferramenta NPM. Posteriormente, foi implementado um cenário de testes com o intuito de verificar o comportamento do sistema ao nível de performance, com base num conjunto de métricas coletadas por uma ferramenta específica de testes em ambientes distribuídos em memória.

Foi possível verificar a importância do conceito *"In-Memory Datagrid"*, na tarefa de armazenar e processar enormes volumes de dados em memória, tendo em consideração a

performance na execução de operações, a escalabilidade na medida de acrescentar mais capacidade ao sistema e a disponibilidade evitando perdas de informação.

Tendo em consideração os resultados principais nos casos de teste avaliados foi verificado que a plataforma Pivotal Gemfire em relação ao Apache Ignite consegue processar um maior número de operações por segundo com valores de latência reduzidos e faz uma melhor gestão da memória disponível com valores baixos de processamento de CPU System utilizado. Para além disto foi possível validar a utilização da política de expiração de dados aquando atinge um período de tempo definido.

O futuro nas organizações passa por arranjar soluções para o crescimento acentuado de dados provenientes de diversas fontes, nesta que é a *"Era da Informação"*. A utilização da memória ao invés do disco é uma solução completamente viável e que estará certamente presente no decorrer dos tempos.

5.1. Trabalho Futuro

Esta dissertação abre várias possibilidades para trabalhos futuros. No que se refere a tecnologias IMDG, a possibilidade de estudar produtos como o Hazelcast e Infinispan parecem ser bons pontos de partida, porque, disponibilizam versões *open source* e têm um conjunto de recursos idênticos aos estudados na dissertação.

Também no estudo desenvolvido ao Pivotal Gemfire e Apache Ignite poderíamos acrescentar algo ao trabalho por forma a torná-lo mais rico. No cenário de avaliação, procedemos à execução de testes de 30 minutos, 1 hora, 2 horas e 3 dias, o que não foi suficiente para levar ao limite a cache de dados, pois a capacidade dos servidores utilizados era enorme. O que poderia ser feito para estudar o comportamento da mesma, levando-a ao limite, seria a execução de testes num prazo de tempo maior, por exemplo 7 ou 15 dias, e com um maior volume de entradas nos servidores de cache. A inclusão de mais nós na topologia seria importante para testar a escalabilidade e performance do sistema à medida que seria preciso mais capacidade de recursos.

Fazer um estudo comportamental do sistema, ao nível dos cenários de avaliação, em parâmetros de configuração *tunning* para diferentes perfis de utilização dos servidores de cache poderia ajudar a encontrar as condições ideias de funcionamento dos mesmos, bem como um estudo comparativo entre a solução IMDG implementada e a solução *Database ORACLE*.

Referências

Abigail, D. & Ribeiro, C., 2013. *Bases de Dados NewSQL : uma avaliação experimental. Projeto de Licenciatura*. Instituto Politécnico de Coimbra.

Alexandre, R.B.R., 2009. *Métricas e Ferramentas Livres para Análise de Capacidade em Servidores Linux. Monografia*. Minas Gerais, Brasil.

Altibase, 2014. Performance comparison against traditional on-disk DBMSs. Available at: http://altibase.com/faq/how-does-altibase-compare-to-other-database-vendors/ [Accessed September 23, 2015].

Apache Ignite, 2015. Cache Modes. Available at: http://apacheignite.gridgain.org/docs/cache-modes [Accessed October 2, 2015].

Barreiro, D., 2014. Dados Estruturados vs Dados Não-Estruturados. Slides de Aula. p.19.

Bondi, A., 2000. Characteristics of Scalability and Their Impact on Performance. Article. *Proceedings of the 2nd international workshop on Software and performance*, pp.195–203.

Borges, R., 2013. Big Data. *UFF Tech 2013*. Available at: http://www.slideshare.net/stiuffmidia/uff-tech-2013-28612300 [Accessed October 15, 2015].

Canaltech, 2014. Big Data: os cinco Vs que todo mundo deveria saber. Available at: http://corporate.canaltech.com.br/dica/big-data/Big-Data-os-cinco-Vs-que-todo-mundo-deveria-saber/ [Accessed October 20, 2015].

D.Schneider, R., 2012. *Hadoop for Dummies*. *Book* Special Ed. J. Wiley & S. Canada, eds., Mississauga.

Demchenko, Y. et al., 2013. Addressing big data issues in Scientific Data Infrastructure. Journal Article. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, (June), pp.48–55.

Dubray, J.-J., 2011. NoSQL e NewSQL. *InfoQueue*. Available at: http://www.infoq.com/br/news/2011/06/newsql [Accessed November 13, 2015].

Gayberi, M. et al., 2013. A study for performance comparison of different in-memory databases. Journal Article. *2013 7th International Conference on Application of Information and Communication Technologies*, pp.1–5.

Gideon, G., 2014. Gartner 2014 CoolVendor. *GridGain*. Available at: http://www.gridgain.com/resources/papers/gartner-cool-vendor/ [Accessed November 13, 2015].

Grolinger, K. et al., 2013. Data management in cloud environments: NoSQL and NewSQL data stores. Journal Article. *Journal of Cloud Computing: Advances, Systems and Applications*, 2, p.22.

Guerreiro, C., 2015. [Infográfico] Big Data em Números. Available at: http://blog.tecnologiaqueinteressa.com/2015/02/infografico-big-data-em-numeros-o-que_2.html [Accessed December 5, 2015].

Gutierrez, D., 2014. GridGain In-Memory Data Fabric. *Inside Big Data*. Available at: http://insidebigdata.com/2014/10/23/gridgain-memory-data-fabric/ [Accessed December 5, 2015].

Hall, A., 2013. key-value store. *Composite Code*. Available at: http://compositecode.com/tag/key-value-store/ [Accessed December 10, 2015].

Informant, 2014. Aprenda a importância da escalabilidade. Available at: http://www.informant.com.br/blog/2014/02/05/aprenda-a-importancia-da-escalabilidade/ [Accessed December 12, 2015].

Intel, 2013. A Internet em um minuto. Available at: http://www.aec.com.br/Site/Noticia/7621?slug=a-internet-em-um-minuto [Accessed November 13, 2015].

Ivanov, N., 2012. In-Memory Compute Grid Explained. *Big Data Zone*. Available at: https://dzone.com/articles/memory-compute-grid-explained [Accessed November 14, 2015].

Jennings, B., 2014. GemFire: A Day in the Life. *Pivotal Labs*. Available at: https://blog.pivotal.io/labs/labs/gemfire_introduction [Accessed November 13, 2015].

Ji, S. et al., 2012. Constructing a data accessing layer for in-memory data grid. Journal Article. *Proceedings of the Fourth Asia-Pacific Symposium on Internetware - Internetware '12*, pp.1–7.

João, D., 2015. *Extensão de Propriedades SQL a SGBD NoSQL através de Call Level Interfaces. Dissertação de Mestrado*. Universidade de Aveiro.

Kumar, G., 2014. Exploring the different types of nosql databases. *3PillarGlobal*. Available at: http://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases [Accessed January 10, 2016].

Lauriano, M., 2005. *Gestão de segurança da informação. Tese de Mestrado.*

Matyashovsky, T., 2013. From cache to in-memory data grid. *SlideShare*. Available at: http://pt.slideshare.net/tmatyashovsky/from-cache-to-in-memory-data-grid-introduction-to-hazelcast [Accessed January 11, 2016].

Network, N.S., 2012. Nokia Siemens Networks Performance Manager. Technical Report. , p.2.

Networks, N., 2014. Performance Manager insights from mountains of data. Technical Report. , p.8.

Networks, N.S. and, 2014. *Product Description Nokia Solutions and Networks Performance Manager. Technical Report.*

Pivotal, 2014. Overview of Pivotal GemFire. *Pivotal Documentation*. Available at: http://gemfirexd.docs.pivotal.io/1.0.0/userguide/about_gemfirexd/topics/c_what_is_elasticsql.html [Accessed January 15, 2016].

Plat, S. & Kekligian, M., 2009. Big Data: O Futuro da Informação e dos Negócios. Artigo de Jornal. *Ene ediciones*.

Plattner, H., 2009. A common database approach for OLTP and OLAP using an in-memory column database. *SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp.1–2. Available at: http://dl.acm.org/citation.cfm?id=1559846&dl=ACM&coll=DL&CFID=573334643&CFTOKEN=16471848.

Prins, R., 2013. In-Memory Data Grids. *Big Data Zone*. Available at: https://dzone.com/articles/memory-data-grids [Accessed January 13, 2016].

Pronschinske, M., 2010. Open Source NoSQL Databases. *DZone*. Available at: https://dzone.com/articles/open-source-nosql-databases [Accessed November 17, 2015].

Rivera, J. & Meulen, R. van der, 2013. Gartner Says In-Memory Computing Is Racing Towards Mainstream Adoption. *Gartner*. Available at: http://www.gartner.com/newsroom/id/2405315 [Accessed November 17, 2015].

Roe, C., 2012. ACID vs BASE. *Data Education for Business and IT Professionals*. Available at: http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-

database-transaction-processing/ [Accessed November 18, 2015].

Schuncke, A., 2013. Comunicação de dados Síncrona x Assíncrona. *Oficina da Net*. Available at: https://www.oficinadanet.com.br/post/9978-comunicacao-de-dados-sincrona-x-assincona [Accessed November 18, 2015].

Setrakyan, D., 2012. In-Memory Data Grids Explained. *Big Data Zone*. Available at: https://dzone.com/articles/memory-data-grids-explained [Accessed November 21, 2015].

Sousa, P., 2010. O teorema CAP. Available at: https://unrealps.wordpress.com/2010/12/28/o-teorema-cap/ [Accessed November 21, 2015].

Souza, A., 2013. NoSQL, Base VS ACID e Teorema CAP. *SlideShare*, p.50. Available at: http://pt.slideshare.net/Celio12/nosql-base-vs-acid-e-teorema-cap [Accessed November 25, 2015].

Souza, A., 2014. Sistemas NoSQL, surgimento, características e exemplos. *SlideShare*, p.15. Available at: http://pt.slideshare.net/Celio12/trabalho-no-sql-aricelio-de-souza [Accessed November 26, 2015].

Systems, G., 2014a. *GridGain vs . Oracle Coherence. Technical Report.*, Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_Oracle_Coherence.html [Accessed January 10, 2016].

Systems, G., 2015a. *GridGain vs. GemFire. Technical Report.*, Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_GemFire.html [Accessed January 10, 2016].

Systems, G., 2014b. *GridGain vs. GigaSpaces. Technical Report.*, Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_GigaSpaces_LP.html [Accessed January 10, 2016].

Systems, G., 2015b. *GridGain vs. Hazelcast.Technical Report.*, Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_Hazelcast_LP.html [Accessed January 10, 2016].

Systems, G., 2014a. GridGain vs. Infinispan. Technical Report. , p.3. Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_Infinispan_LP.html [Accessed January 10, 2016].

Systems, G., 2014b. *GridGain vs. Terracotta. Technical Report.*, Available at: http://go.gridgain.com/Feature_Comparison_GridGain_vs_Terracotta_LP.html [Accessed January 10, 2016].

TayzGrid, 2015. *GridGain vs . TayzGrid. Technical Report.*, Available at: http://www.alachisoft.com/resources/comparisons/gridgain-vs-tayzgrid.html

[Accessed January 10, 2016].
Technologies, E., 2015. What is MRAM? *Future Electronics*. Available at: http://www.futureelectronics.com/en/memory/mram.aspx [Accessed December 20, 2015].

Tel, U.M., 2016. *ETSI TS 1 UTRAN network resources. Technical Report*.

Vamsi, B. & Venkatesh, P., 2012. A importância de bancos de dados in-memory. *OpenSourceForU*. Available at: http://opensourceforu.efytimes.com/2012/01/importance-of-in-memory-databases/.

Wikipédia, 2013a. Computação de alto desempenho. Available at: https://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_de_alto_desempenho [Accessed December 20, 2015].

Wikipédia, 2013b. Sistema de alta disponibilidade. Available at: https://pt.wikipedia.org/wiki/Sistema_de_alta_disponibilidade [Accessed December 20, 2015].

Williams, J.W. et al., 2014. *Bridging High Velocity and High Volume Industrial Big Data Through Distributed In-Memory Storage & Analytics. Journal Article.*,

Apêndices

Apêndice A

Prova de Conceito

#Ficheiro "config.properties" Servidor e Cliente

```
prop.serverAddress=10.92.116.160:47500..47509,10.92.116.161:47500..47509,10.92.116.162:47500..47509
```

#Configuração Cliente

```
public IgniteConfiguration configurationClient() {
    //Load File Properties
    prop = getFileProperties();
    //1-Modo Cliente
    Ignition.setClientMode(true);

    TcpDiscoverySpi spi = new TcpDiscoverySpi();
    TcpDiscoveryVmIpFinder ipFinder = new TcpDiscoveryVmIpFinder();
    // Set initial IP addresses.
    String stringAddress = prop.getProperty("prop.serverAddress");
    String arrayAddress[] = stringAddress.split(",");
    ipFinder.setAddresses(Arrays.asList(arrayAddress));
    spi.setIpFinder(ipFinder);

    IgniteConfiguration cfg = new IgniteConfiguration();
    // Override default discovery SPI.
    cfg.setDiscoverySpi(spi);

    return cfg;
}
```

#Configuração Servidor

```
public void configurationServer() {
    //Load File Properties
    prop = getFileProperties();

    TcpDiscoverySpi spi = new TcpDiscoverySpi();
    TcpDiscoveryVmIpFinder ipFinder = new TcpDiscoveryVmIpFinder();
    // Set initial IP addresses.
    String stringAddress = prop.getProperty("prop.serverAddress");
    String arrayAddress[] = stringAddress.split(",");
    ipFinder.setAddresses(Arrays.asList(arrayAddress));
    spi.setIpFinder(ipFinder);

    IgniteConfiguration cfg = new IgniteConfiguration();

    // Override default discovery SPI.
    cfg.setDiscoverySpi(spi);


    // Start Ignite node.
    Ignition.start(cfg);
}
```

Listagem 1 – Configuração Cliente e Cache de Servidor Apache Ignite/GridGain.

#Configuração de Cache

```
private static final String NAME_CACHE = "reportCacheNPM";
private static IgniteCache<String, List<ObjectValue>> reportCache;

public void configurationCache(IgniteConfiguration cfg) {
    //******************************************************************
    //Initialize Ignite
    org.apache.ignite.Ignite ignite = Ignition.start(cfg);
    //******************************************************************
    //1-Configuração de Cache
    CacheConfiguration<String, List<ObjectValue>> objectCfg = new CacheConfiguration<>(NAME_CACHE);
    //******************************************************************
    //1-Modo de Atomicidade
    objectCfg.setAtomicityMode(CacheAtomicityMode.ATOMIC);
    //2-Modo de Cache (Partitionated or Replicaded in nodes)
    objectCfg.setCacheMode(CacheMode.PARTITIONED);
    //3-Definir Backups da Cache
    objectCfg.setBackups(1);
    //6-Modos de Memória
    objectCfg.setMemoryMode(CacheMemoryMode.ONHEAP_TIERED);
    //7-Politicas de Esvaziamento (Eviction Policies) da Cache em memória On-Heap e Off-Heap aquand
    objectCfg.setEvictionPolicy(new LruEvictionPolicy(1000000));
    //9-Politicas de Expiração de Dados na cache
    objectCfg.setExpiryPolicyFactory(CreatedExpiryPolicy.
            factoryOf(new javax.cache.expiry.Duration(TimeUnit.DAYS, 3)));
    cfg.setCacheConfiguration(objectCfg);
    // Create cache with given name, if it does not exist.
    reportCache = ignite.getOrCreateCache(objectCfg);
}
```

Listagem 2 – Configuração Cache Apache Ignite/GridGain.

#Ficheiro "config.properties" Mediador

```
#Config Nokia
prop.locatorAddress=10.92.116.168
prop.locatorPort=13491
prop.locatorName=locator1
```

#Configuração Mediador

```
  private final LocatorLauncher locatorLauncher;
  public startLocator() {
      //Load File Properties
      prop = getFileProperties();
      String addressLocator = prop.getProperty("prop.locatorAddress");
      int portLocator = Integer.valueOf(prop.getProperty("prop.locatorPort"));
      String nameLocator = prop.getProperty("prop.locatorName");

      locatorLauncher = new LocatorLauncher.Builder()
              .set("log-file", "gemfireLOG.log")
              .set("jmx-manager", "true")
              .set("jmx-manager-start", "true")
              .set("jmx-manager-http-port", "7070")
              .set("jmx-manager-ssl", "false")
              .setMemberName(nameLocator)
              .setPort(portLocator)
              .setBindAddress(addressLocator)
              .setRedirectOutput(true)
              .build();
  }
```

```
public void run() {
    System.out.println("Attempting to start Locator");
    // start the Locator in-process
    locatorLauncher.start();

    // wait for Locator to start and be ready to accept member (client) conn
    locatorLauncher.waitOnStatusResponse(300, 300, TimeUnit.MINUTES);
    System.out.println("Locator successfully started");
}
```

#Ficheiro "config.properties" Servidor

```
#Config Nokia
prop.ServerName=server1
prop.ServerPort=13493
prop.locatorConnect=10.92.116.168[13491]
prop.fileCacheConfig=/cache.xml
```

#Configuração Servidor

```
public static void main(String[] args) throws URISyntaxException {
    //Load File Properties
    prop = getFileProperties();
    int portServer = Integer.valueOf(prop.getProperty("prop.ServerPort"));
    String connectLocator = prop.getProperty("prop.locatorConnect");
    String nameServer = prop.getProperty("prop.ServerName");
    String fileCacheConfig = prop.getProperty("prop.fileCacheConfig");
    path = startServer3.class.getProtectionDomain()
            .getCodeSource().getLocation().toURI().toString();
    path = path.substring(0, path.lastIndexOf("/")).replace("file:", "");
    String x = path + fileCacheConfig;

    ServerLauncher serverLauncher = new ServerLauncher.Builder()
            .setMemberName(nameServer)
            .setServerPort(portServer)
            .set("locators", connectLocator)
            .set("cache-xml-file", x)
            .set("log-level", "info")
            .set("log-file", "gemfireLOG.log")
            .build();

    System.out.println("Attempting to start cache server");
    serverLauncher.start();
    System.out.println("Cache server successfully started");
}
```

#Ficheiro "config.properties" Cliente

```
#Config Nokia
prop.locatorAddress=10.92.116.168
prop.locatorPort=13491
```

#Configuração Cliente

```
private static ClientCache cache;
public void ConfigurationClient() {
    //Load File Properties
    prop = getFileProperties();
    String addressLocator = prop.getProperty("prop.locatorAddress");
    int portLocator = Integer.valueOf(prop.getProperty("prop.locatorPort"));
    cache = new ClientCacheFactory()
            .addPoolLocator(addressLocator, portLocator)
            .setPoolSubscriptionEnabled(true)
            .create();
}
```

Listagem 3 – Configuração Servidores de Cache, Localizador e Cliente Apache Geode/Pivotal Gemfire.

#Configuração de Cache Local

```
//File Config Local Cache
private static Region<String, List<ObjectValue>> reportCache;

@Override
public void ConfigurationCache() {
    reportCache = cache.<String, List<ObjectValue>>createClientRegionFactory(
            ClientRegionShortcut.CACHING_PROXY_HEAP_LRU)
            .create("reportCache");
}
```

#Configuração de Cache Servidor

```
<!--File Config Servers Cache cache.xml-->
<cache
    xmlns="http://schema.pivotal.io/gemfire/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schema.pivotal.io/gemfire/cache http://schema.
    version="8.1">
    <cache-server port="13493"/>
    <region name="reportCache">
        <region-attributes data-policy="partition" statistics-enabled="true">
            <entry-time-to-live>
                <expiration-attributes timeout="259200" action="destroy"/>
            </entry-time-to-live>
            <partition-attributes redundant-copies="1"/>
            <eviction-attributes>
                <lru-entry-count maximum="1000000" action="local-destroy"/>
            </eviction-attributes>
        </region-attributes>
    </region>
</cache>
```

Listagem 4 – Configuração Cache Local e Servidores de Cache Apache Geode/Pivotal Gemfire.

#Método setUp(...)

```
private static final IGemFire gemfire = new GemFire();
private static Properties prop = new Properties();
private static List<String> queryArray = new CopyOnWriteArrayList<>();
private static String reportKey;
private final HashMap<String, String> reportParameters = new HashMap<>();

@Override
public void setUp(BenchmarkConfiguration cfg) throws Exception {
    super.setUp(cfg);
    println(cfg, "Read Querys and Connection JDBC...");

    long start = System.nanoTime();
    //Load das querys do ficheiro e armazenamento em uma lista de querys
    prop = gemfire.getFileProperties();
    String stringQuerysFile1 = prop.getProperty("prop.nameFileQueries");
    queryArray = LoadXMLQuerys.readForLinesXML(stringQuerysFile1);
    for (String querys : queryArray) {
        String[] parts = querys.split("#");
        //Chave Query
        reportKey = parts[0] + "#" + parts[1] + "#" + parts[2] + "#" + parts[3] + "#" + parts[4];
        //Valor Query
        String query = parts[5];
        reportParameters.put(reportKey, query);
    }
    System.out.println("Numero de Queries -->" + reportParameters.size());
    println(cfg, "Finished Read Querys and Connection in "
            + ((System.nanoTime() - start) / 1_000_000) + " ms.");
}
```

#Método test(...)

```
public boolean test(Map<Object, Object> ctx) throws Exception {
    Connection conn = null;
    //Random to HashMap reportParameters
    Object[] keys = reportParameters.keySet().toArray();
    String keyRandom = (String) keys[ThreadLocalRandom.current().nextInt(keys.length)];

    //GET EndTime
    Date now = new Date();
    Calendar cal = Calendar.getInstance();
    String startTimestamp = null;
    SimpleDateFormat formatter1 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    Date endTime1 = DateUtils.round(now, Calendar.HOUR);
    cal.setTime(endTime1);
    cal.add(Calendar.HOUR, -1);
    Date endTime = cal.getTime();
    String endTimestamp = formatter1.format(endTime);

    //GET StartTime
    if (keyRandom.contains("Hour")) {
        cal.setTime(endTime);
        cal.add(Calendar.HOUR, -1);
        Date startTime = cal.getTime();
        startTimestamp = formatter1.format(startTime);
    } else {
        cal.setTime(endTime);
        cal.add(Calendar.HOUR, -1);
        cal.add(Calendar.DAY_OF_MONTH, -1);
        Date startTime = cal.getTime();
        startTimestamp = formatter1.format(startTime);
    }
```

```
    //Replace Key
    String key = keyRandom.replace("startParam", startTimestamp).replace("endParam", endTimestamp);
    //Replace Query
    String query1 = reportParameters.get(keyRandom);
    String query = query1.replace("startParam", startTimestamp).replace("endParam", endTimestamp);

    //GET a Chave Random do HashMap
    Object val = getKeyReport(key);
    if (val != null) {
        System.out.println("*****************************************************");
        System.out.println("Chave-->" + key + "\n");
        System.out.println("Valores-->" + val + "\n");
        System.out.println("*****************************************************");
    } else {

        //PUT REPORT
        System.out.println("*****************************************************");
        System.out.println("NÃ£o tem dados na cache! Armazenamento");
        //InicializaÃ§Ã£o das conexÃµes JDBC
        System.out.println("Connecting to database...");

        conn = DriverManager.getConnection(gemfire.getDB_URL(), gemfire.getUser(), gemfire.getPass());
        Class.forName(gemfire.DRIVER());
        storeResultsQueries(key, query, conn);
        conn.close();
        System.out.println("Add Report: " + testRegion.size());
        System.out.println("*****************************************************");
    }
    return true;
}
```

```
public Object getKeyReport(Object key) {
    Object val = testRegion.get(key);
    return val;
}

public void storeResultsQueries(String reportKey, String query, Connection conn)
        throws ClassNotFoundException, SQLException {
    //Inicialização de variáveis locais
    Statement stmtResults;
    ResultSet resResults;
    List<ObjectValue> obj = new CopyOnWriteArrayList<>();
    List<String> listColumnsKPI = new CopyOnWriteArrayList<>();
    List<Double> listKPI = new CopyOnWriteArrayList<>();
    String id_object = null;
    Timestamp period_start_time = null;
    String sql;
    int total = 0;

    //STEP 4: Execute Query
    stmtResults = conn.createStatement();

    sql = query;
    resResults = stmtResults.executeQuery(sql);
    ResultSetMetaData metadata = resResults.getMetaData();

    int numberOfColumns = metadata.getColumnCount();

    obj.clear();
    if (resResults.next()) {
```

```
do {

    //Clean Lista de Colunas e KPIs
    listColumnsKPI.clear();
    listKPI.clear();
    //*****************************************************************
    for (int i = 1; i < numberOfColumns + 1; i++) {
        int type = metadata.getColumnType(i);
        //Nokia=DECIMAL
        if (type == Types.DECIMAL) {
            String columnName = metadata.getColumnName(i);
            listColumnsKPI.add(columnName);
            listKPI.add(resResults.getDouble(columnName));
        }
        if (type == Types.TIMESTAMP) {
            period_start_time = resResults.getTimestamp(i);
        }
        if (type == Types.VARCHAR) {
            String columnName = metadata.getColumnName(i);
            //Get Object Query of id_Object
            String[] parts = reportKey.split("#");
            String[] arr = parts[0].split("/");
            String object = arr[arr.length - 1];
            object = object.replaceAll("-\\d", "").toLowerCase();
            if (columnName.contains(object)) {
                id_object = resResults.getString(i);
            }
        }
    }
    obj.add(new ObjectValue(total, id_object, period_start_time, listColumnsKPI, listKPI));
                total++;
                //*************************************************************

            } while (resResults.next());
        }
        //STEP 6: Close connection
        if (!obj.isEmpty()) {
            testRegion.put(reportKey, obj);
        }
        stmtResults.close();
    }
}
```

Listagem 5 – Classe PUTGETXMLBenchmark.java